uc3m | Universidad Carlos III de Madrid

Desarrollo de Técnicas de Inteligencia Artificial Aplicadas a los Juegos de Atari

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática

24 de septiembre de 2018

Autor: Jorge Carmena Plaza

Tutor: Daniel Borrajo Millán

Agradecimientos

Quiero agradecer a mi familia el apoyo, sacrificio y comprensión que me han mostrado durante estos cuatro largos años:

A mi padre, Félix, por su disposición a llevarme y recogerme de la estación de tren a diario, por escucharme cuando quería expresar mi alegría, mi enfado o mi tristeza, y por respetar mi silencio cuando no estaba de humor. También por sus frases motivadoras cuando tenía exámenes, y por ser un ejemplo de perseverancia y de valentía.

A mi madre, Carolina, por sus sabios consejos, su amor incondicional y su paciencia infinita. También por ayudarme a restarle importancia a situaciones que me disgustaban excesivamente, y por su enorme dedicación al atender las labores del hogar que me correspondían para que pudiera emplear todo mi tiempo a los estudios.

A mi hermana, Andrea, por estar ahí siempre que la he necesitado, por llegar a casa siempre dispuesta a conversar alegremente, haciendo que olvide por un momento mis preocupaciones.

A mis abuelos, tíos y primos, por mostrar su interés, por apoyarme con entusiasmo y por darme ánimos en las reuniones familiares.

También quisiera agradecer a mis compañeros por trabajar incansablemente a mi lado. Con ellos he compartido momentos de frustración y tensión, pero también de felicidad y risas. Con ellos he compartido mesa y ordenador, he pasado la noche conectado a Skype para terminar algunas prácticas y me he sentado en el suelo de la estación de tren para entregar otras tantas.

Por último, me gustaría agradecer a los profesores por compartir con nosotros sus conocimientos, por motivarnos para alcanzar nuestras metas y por convertir nuestro paso por la universidad en una experiencia grata y provechosa.

Sin ninguno de vosotros jamás podría haber llegado hasta aquí.

Gracias.

Índice

Índice de Ilustraciones

Índice de Tablas

1 Resumen

Arcade Learning Environment (ALE) es una popular plataforma que facilita a los investigadores el desarrollo de agentes inteligentes para resolver los videojuegos de Atari. Los estudios con la plataforma han demostrado la utilidad de diversas técnicas de Inteligencia Artificial (IA) para la toma de decisiones en el dominio de Atari. Sin embargo, existe un amplio margen de mejora, puesto que hasta el momento ningún agente ha logrado abordar los videojuegos de manera perfecta. Por esta razón, en el presente trabajo se propone crear un nuevo agente basado en la colaboración de dos reconocidas técnicas de IA: *Deep Q-Learning* (DQN) e *Iterated Width* (IW). DQN es un algoritmo de aprendizaje por refuerzo que entrena una red convolucional para que tome decisiones basándose en imágenes del estado actual de la partida, mientras que IW es un algoritmo de planificación que realiza simulaciones del futuro de la partida para tomar decisiones.

La creación del nuevo agente está guiada por una metodología de desarrollo *software* que divide el proceso en etapas secuenciales: análisis, diseño, implementación, pruebas, verificación y mantenimiento. El nuevo agente se construye tomando como base el *framework* ALE y aprovechando las implementaciones de los algoritmos IW y DQN creadas por diversos investigadores del dominio de Atari.

El proyecto incluye una extensa evaluación para medir la calidad del nuevo agente en un abanico de videojuegos de Atari. De la experimentación puede concluirse que el nuevo agente se postula como una prometedora alternativa a los agentes tradicionales del dominio, ya que supera los resultados obtenidos en una parte de los videojuegos probados, y es competente en aquellos videojuegos en los que no obtiene las mejores puntuaciones. Aun así, sería posible mejorar los resultados si se superasen las restricciones temporales y de *hardware* impuestas necesariamente a lo largo de la experimentación, y que afectan principalmente a la calidad del entrenamiento de las redes convolucionales con DQN.

2 Abstract

2.1 Introduction

In this document, the developed Bachelor Thesis is exposed. It is an experimental project involving different branches of Artificial Intelligence to solve issues related to the entertainment industry, specifically, in the field of video games. The project is based on recognized research in the sphere of Artificial Intelligence, applies concepts from other computing sectors such as Software Engineering, and includes an exhaustive experimentation that totals more than 300 hours of execution.

The *Atari 2600* console has many games that, at present, can be executed by a multiplatform emulator called *Stella*. Based on this emulator, several researchers have developed a software called *Arcade Learning Environment* (ALE) that opens doors to the study of Artificial Intelligence techniques on the video games of that console. The goal of the researchers is to build agents that can solve games exhibiting an intelligent behavior, that is, choosing wisely the best action in each state of the game.

Currently, there are three main approaches to solving the Atari games: *reinforcement learning*, *planning* and *genetic programming*. Each one of them has its advantages and disadvantages, and they are not perfect. The **main objective** of this work is to develop a new agent based on Artificial Intelligence techniques that can make decisions in real time in dynamic environments such as Atari videogames.

To guide the project, some sub-objectives have been marked:

1. Study of the current state of the project topic, including theoretical basis, Artificial Intelligence techniques developed so far, and software tools used by researchers.
2. Construction of a new agent that combines different techniques used in the field of Atari games (*reinforcement learning* and *planning*), following a software development methodology that ensures the correct analysis, design and implementation of the agent.
3. Experimentation to test the new agent in multiple Atari video games. It will be exhaustive but adapted to the limitations of the portable computer that will be used. Its performance is greatly exceeded by the machines used in the field of Atari video games. Therefore, the limitations of the experiments will be established in a clear and justified way.
4. Analysis of the results obtained in the experimentation to find out how the new agent behaves, in comparison with other agents studied in the state of the art.

2.2 State of the art

2.2.1 Theoretical Framework

Artificial Intelligence (AI) is a branch of Computer Science dedicated to developing techniques that allow computers to solve problems that require the learning or reasoning capabilities of human beings.

Companies have recognized the great utility of Artificial Intelligence to improve their products or increase the efficiency and effectiveness of their business model. There are many sectors that make use of Artificial Intelligence: customer service, finance and banking, marketing, medicine and robotics, among many others.

The video game sector stands out for having incorporated many AI techniques recently with the aim of improving the quality of video games: that they are more dynamic and imaginative, that they do not quickly bore the player and represent a challenge for them.

The AI subfields that are commonly used in this area are the following:

- **Machine learning** [1]**.** It gathers a group of techniques that allow a software system to learn to solve specific tasks. To properly tackle the task, the system must be trained by providing a set of training examples. Reinforcement learning [2] is a type of learning specialized in tasks in which an agent interacts in discrete time with an environment to reach a final state from an initial one, maximizing the reinforcement received.

- **Artificial neural networks** [3]**.** They are bio-inspired computational models that are part of the machine learning techniques. It is a set of interconnected neurons that receive information as input and process it to specify an output. Recently, the concept *Deep Learning* [4] has become popular. It is a machine learning subfield that uses artificial neural networks with multiple layers (like convolutional neural networks [5]) to deal with tasks that require a large volume of data, such as image recognition, video classification, natural language processing, etc.

- **Search** [6]**.** It includes those techniques useful for solving problems that can be defined as a state space and a set of actions. The objective is to find a sequence of actions that reach a goal state from an initial one.

- **Evolutionary computation** [7]**.** It includes those techniques of Artificial Intelligence capable of solving problems emulating the principles of biological evolution. It is especially useful for problems in which finding the optimal solution is not possible. The objective is to evolve a population of solutions in successive generations to obtain ever better solutions to the problem. Genetic programming is an evolutionary computation approach in which the solution to a problem is a computer program.

Atari [8] [9] is an American producer considered to be the initiator of the videogame industry. Its *Atari 2600* console was a success and popularized the concept of interchangeable cartridges, which laid the foundation for future generations of video game consoles. Thanks to the removable cartridges, the console came to have a wide video games catalog.

2.2.2 Software tools

The tools commonly related to the project topic are the following:

- ***Arcade Learning Environment*** (ALE) [10] is a framework that allows researchers to develop intelligent agents on more than 50 Atari video games, using Artificial Intelligence techniques such as reinforcement learning or search/planning.

  ALE is built on *Stella*, an emulator of *Atari 2600*. It serves the developer as an interface to interact with the environment, so he does not have to worry about emulation details.

- ***Gym*** [11] is a framework that facilitates the development and evaluation of reinforcement learning techniques. It is a library that gathers a collection of environments, including *Atari*. This environment integrates the ALE framework, so that it allows to emulate many *Atari 2600* video games.

- **Scientific computing libraries.** They provide tools to develop systems that, among other things, learn to solve machine learning and deep learning tasks. *TensorFlow* [12] and *PyTorch* [13] stand out among the many alternatives.

2.2.3 Case studies

2.2.3.1 *Playing Atari with Deep Reinforcement Learning*

This article [14], published in 2013 by *DeepMind*, formulates and evaluates an algorithm that relates *deep learning* and *reinforcement learning* concepts: *Deep Q-Learning* (DQN).

The intention of DQN is to train intelligent agents for environments whose state space is extremely large. For this type of environment, traditional reinforcement learning algorithms (*Q-Learning*, for example) are ineffective.

In DQN, the agent is a convolutional neural network that receives as inputs the state of the environment in the form of raw images. Training the agent is like training the convolutional network so that, given a state represented with images, the network determines the best action.

The objective of *DeepMind* was to train agents with DQN to solve the *Atari 2600* video games, using only RGB video outputs, reinforcements and termination signals provided by the ALE framework [15].

The experimentation of DQN was done on a subset of *Atari 2600* video games: *Beam Rider*, *Breakout*, *Enduro*, *Pong*, *Q* Bert*, *Seaquest* and *Space Invaders*. For all of them, the architecture of DQN and its parameters were not modified. The authors also used other learning algorithms to compare the results.

The results determined that DQN had achieved the best score for 6 of the 7 tested video games. This demonstrated that the combination of convolutional neural networks with reinforcement learning is feasible to train agents from large volumes of sensory data of high dimensionality.

2.2.3.2 *Human-level control through deep reinforcement learning*

The authors wrote another experimental paper [16] in which they tested DQN on a much larger set of Atari video games. They had the same considerations as in the other article, but they changed the architecture of the convolutional network a little.

They applied the same architecture and the same DQN parameters for all video games of experimentation and obtained very positive results in a large part of them.

2.2.3.3 *Classical Planning with Simulators: Results on the Atari Video Games*

In this article [17] a planning algorithm is evaluated to solve Atari video games using the ALE framework proposed by Bellemare and his team in their article of 2013 [15].

ALE allows the researchers to tackle the task in two ways: with *planning* and with r*einforcement learning*. The authors of the article focused on the study of planning techniques and left aside those of reinforcement learning.

The problem of using classic planners in ALE is that there is no domain coding in the PDDL style and the goal to be achieved is not known either. Therefore, the authors focused on new planning algorithms, related to non-informed search methods. These algorithms can work efficiently in large states spaces without the need to know goals, costs or transitions between states. IW is the planning algorithm evaluated in the article.

IW is based on a blind breadth first search and introduces the concept of *novelty* to prune unnecessary nodes. The greater the *novelty* value of the node, the older it is.

IW was compared in time and average score with other planning algorithms on 54 *Atari 2600* video games. The results of the experimentation determined that IW obtained the best average score in 27 of the 54 games, and that it obtained the shortest execution time in 18 of the 54 games.

2.2.3.4 *Evolving simple programs for playing Atari games*

It is a very recent article [18] in which a method called *Cartesian Genetic Programming* (CGP) is proposed to tackle the task of solving Atari video games.

CGP is an evolutionary computing technique that has proven to be very useful to solve image processing or filtering tasks in various fields. Given that the framework ALE [15] facilitates the development of agents based on AI techniques and that provides the state of the Atari games in the form of pixel arrays, the researchers considered that CGP could fit in the Atari video games domain.

The objective of the researchers was to generate computer programs that solve Atari games by making decisions based on the processing of input pixel arrays.

CGP is a form of genetic programming in which the individuals of the population represent computer programs through directed graphs and, normally, acyclic. The nodes of the graph are connected by means of Cartesian coordinates. In addition, each node has an associated function, usually mathematical or statistical (ADD, STDDEV, FLOOR, etc.), which receives as input the output data of the nodes that are connected to it.

The evolution of an individual involves a change in the functions associated with the nodes and in the connections between them, so the program encoded in the individual changes the way to process the inputs.

The researchers experimented with CGP on 61 video games, and the results were compared with other techniques applied so far in the Atari domain. In 9 of the video games CGP obtained the highest score, which was quite an achievement considering that this work represented a first approximation and that the evolved programs were relatively simple. In the other games, CGP provided improvable but promising results.

CGP is consolidated as a competent technique in Atari domain. The main advantage of CGP is the simplicity of the programs and the speed with which they evolve. However, it still needs to improve in some respects.

2.3 Presentation of the issue

As we have seen in the articles analyzed in the case studies about DQN [14] and IW [17], there are two consolidated lines of research in the domain of Atari video games: *planning* and *reinforcement learning*. Each approach has its own advantages and disadvantages, and they are not perfect.

The question that arises in this project is whether it is possible to combine these two approaches to improve the way of solving the games of the *Atari 2600* console. It is proposed to use a trained convolutional network with the DQN algorithm (from the case study [16]) to help the IW algorithm (from the case study [17]) to decide which actions are most appropriate throughout the game. From now on, this approach will be called ***IW+DQN combination***.

2.4 Development of the issue

In this section, the IW+DQN combination proposed in the previous section will be developed. For this, a software development methodology will be necessary. It will be based on the *cascade model* [19], which divides the software project into stages that are executed sequentially following a logical order. The common stages are the following:

1. **Analysis.** The objectives are established and the needs that software must cover are analyzed, formalizing them as requirements.
2. **Design.** Considering the previous analysis, the architecture and theoretical behavior of the software are defined by means of classes and sequence diagrams.
3. **Coding.** The software is implemented following the design specified in the previous stage.
4. **Testing.** With the software already implemented, it is verified that it works correctly and that it meets the requirements.
5. **Checking.** The software product already tested is delivered to the final user, who executes it and gives its approval.
6. **Maintenance.** Errors that may arise after the delivery of the product to the end user are corrected.

This method has been chosen for three reasons: (1) it is known and easy to understand, (2) it is very suitable for simple projects and in which there is no risk of modifying the initially proposed objectives, and (3) the division by sequential stages facilitates project planning.

2.4.1 Analysis

To develop the IW+DQN combination, two existing codes were taken as a basis:

- The *C++* code extracted from ***ALE-Atari-Width*** repository of *Github* [20]. It implements the IW planning algorithm. It was constructed and used by the authors of the case study about IW to experiment with the algorithm.
- The *Python* code extracted from ***DQN-tensorflow*** repository of *Github* [21], which implements the DQN algorithm using *Gym* [11] and *Tensorflow* [12] libraries. This code has been chosen among many other DQN implementations because it is the most faithful to the specifications of *DeepMind* team in its article about DQN [14].

In this analysis stage, a study of the functionalities offered by *ALE-Atari-Width* code is carried out to specify the objectives of this software development project. The user chooses, among other things, the title of the video game (*game* parameter), the type of game controller (*game_controller* parameter) and the type of agent (*player_agent*

parameter). When the code is executed, the controller is initialized, which is responsible for loading the video game into the emulator and starting the chosen agent. Then, the controller initiates an iterative process in which the game state is received and calls the agent to decide the action to be executed, given that state.

There are several types of agents: *RandomAgent*, *SearchAgent*, etc. Developing the IW+DQN combination proposed is synonymous with developing a new agent, which can be implemented within the *ALE-Atari-Width* code together with the existing ones. This new agent will be called ***CombinationAgent***.

This section includes the user and software requirements that specify all the characteristics of the new agent and its integration with existing software systems.

2.4.2 Design

This section talks about the design of the *CombinationAgent* agent. The design of *ALE-Atari-Width* will be presented at a very high-level because an important part of the agent will be implemented in it, and it is necessary to understand its operation and architecture. For this, class and sequence diagrams will be attached (see Ilustración 17):

The most relevant classes of the *ALE-Atari-Width* system could be grouped as follows:

- **Classes inherited from *ALEController* class.** They implement different types of game controller. The *InternalController* controller is used primarily.
- **Classes inherited from *PlayerAgent* class.** They implement different types of agents. The agent developed by the authors of the study case about IW [17] is *SearchAgent*.
- **Classes inherited from *SearchTree* class.** They implement different types of planning algorithms (including IW, of the *IW1Search* class), which would be used by the *SearchAgent* agent. To develop the simulation tree, all algorithms need the "node" implementation of the *TreeNode* class.

The new classes are *CNNController* and *CombinationAgent*. They are described below:

- ***CNNController*.** This class incorporates all the necessary tools to interact with the convolutional network implemented in the *DQN-tensorflow* code. It focuses on treating network inputs, sending them, calling the network, receiving the outputs and processing them.
- ***CombinationAgent*.** This class implements a new agent, which combines DQN and planning algorithms. Therefore, the class includes an instance of the controller of the DQN network and an instance of the chosen planning algorithm (IW, for example). The class contains the necessary mechanisms to combine both techniques during a game of a video game.

  This class inherits from *PlayerAgent* class.

Other classes of the *DQN-tensorflow* and *ALE-Atari-Width* subsystems have been slightly modified to adapt them to the new agent.

2.5 Evaluation

Once the software system is built, it is evaluated through an experimentation, which is divided into 3 stages. The results of each stage will be used to make decisions about the later stages.

They are explained below:

- **Stage 1.**

  Experimentation with DQN is carried out to fine-tune the algorithm for the next stage. Using a reference video game and starting from an initial parameter configuration extracted from the case study about DQN [16], possible improvements are proposed and tested. These improvements can be value changes of DQN parameters or slight changes in the algorithm implementation. With each experiment, learning evolution graphs of the convolutional network are included, which serve to determine the quality of training.

  Due to the insufficient capacity of the RAM memory of the computer used for the experimentation, the memory used to train the networks with DQN algorithm was limited. A temporary training limit was also established due to the lack of a compatible graphic processor that accelerates the execution.

  At the end of this stage, the code had been modified slightly in relation to the processing of the images that are passed as input to the network.

- **Stage 2.**

  With DQN configured in the previous stage, it is proposed to use the algorithm to train the convolutional networks on a subset of Atari video games. As in stage 1, for each game, various learning evolution graphs of the network will be attached.

  Six video games of *Atari 2600* console were chosen based on the results obtained by the researchers in the case studies about IW and DQN. For the experimentation to be rigorous, different videogame profiles were searched. The games are *Breakout*, *Space Invaders*, *Beam Rider*, *Frostbite*, *Q* Bert* and *Star Gunner*.

- **Stage 3.**

  Once networks have been trained, it is proposed to experiment with the new agent for each video game chosen in the previous stage. Stage 3 is very exhaustive and consists of 2 phases.

In phase 1, *SearchAgent* agent was executed using IW as a planning algorithm, as was done by the authors of case study about IW [17]. For each game, different experiments were carried out, changing the value of the *limite_simulacion* parameter, which regulates the completeness of the simulations. To avoid that the experiments last too long, a limit of 8000 frames per game was established.

In phase 2, *CombinationAgent* agent was executed, also using IW as a planning algorithm. The values 1000, 5000 and 25000 were chosen for parameter *limite_simulacion* because in phase 1 it was observed that they generally provided high scores with less exhaustive simulations and with a reduced time cost. For this experimentation, two *modes* of IW and DQN combination were tested: *Consensus* and *Probability*. The values of parameter *ratio* were also varied. This *CombinationAgent* parameter defines the percentage of influence of the DQN network on IW decisions during the game.

2.6 Analysis of results

2.6.1 Stage 1

With the experiments carried out in this stage it can be guaranteed that the implementation of DQN is able to train intelligent agents. The learning evolution graphs show, at least in *Breakout*, the training allows the agent to learn a strategy within the game to get the highest number of points.

Although the experimentation of this stage has not been completely exhaustive due to hardware limitations, the results are more than acceptable for subsequent stages.

2.6.2 Stage 2

Networks trained with DQN solve games with higher score than a random agent. This shows that DQN has managed to train an intelligent agent for all chosen video games.

However, the training with DQN in this experimentation has not been as productive as the training described in the case study about DQN [16]. The scores obtained in the article are visibly higher than those of this experimentation. Even so, networks will be useful for stage 3 of experimentation.

Some networks stagnate at some point of the training. It is especially evident in the learning evolution graphs of *Breakout* and *Frostbite*, where a maximum is reached too soon, and training is not able to solve. There is also an early halting in *Beam Rider*, but it seems to be solved in the final phase of training. Therefore, it is possible that only more time is needed for the convolutional networks to come out of halting and continue learning.

2.6.3 Stage 3

Against all odds, *SearchAgent* with IW as a planning algorithm works better than expected with small *limite_simulacion* values, which means that IW can solve games with high scores without carrying out extensive and heavy future simulations, which would need a long time.

It is observed that the scores of *Probability* mode are generally lower than those of *Consensus* mode. *Consensus* mode seems more effective than *Probability*, which would mean that it is preferable that the actions be consensual between IW and the convolutional network.

SearchAgent performs better than *CombinationAgent* for *Breakout*, *Space Invaders* and *Beam Rider* games. However, the best scores of the IW+DQN combination are not much below the IW scores so the new agent is a good alternative.

CombinationAgent performs better than *SearchAgent* for *Frostbite*, *Q* Bert* and *Star Gunner* games. In these last two video games, *CombinationAgent* gets to multiply by 7 and by 4 the best scores of *SearchAgent*. For these 3 video games, *CombinationAgent* is also better than the DQN networks trained in stage 2 of the experimentation.

The average time required to execute a frame does not vary between *SearchAgent* and *CombinationAgent* because the time spent per frame in using convolutional networks is insignificant compared to the time spent per frame in the IW simulation, which is the heaviest part of the execution.

Comparing the results of this experimentation with the results of the case studies is not very relevant since the experiments have been carried out in different conditions due to the hardware and time limitations mentioned throughout the experimentation. The results of the case studies served only as a reference to determine which video games could benefit from the new *CombinationAgent* agent. The comparisons between phases 1 and 2 are relevant, because the experiments were carried out under the same conditions.

2.7 Conclusions

The results of the experimentation suggest that the new agent is a good alternative to traditional approaches but should not be a substitute for them. There are more than 50 video games in the Atari domain, so hardly one technique will stand out from the rest in all of them. The important thing is that the agent is competent in as many situations as possible. If we compare the results of *CombinationAgent* and *SearchAgent* under the same experimental conditions, the new agent seems competent even in the video games in which it is surpassed by *SearchAgent*, since the score differences in these worst cases are very small.

The collaboration of two disparate techniques such as DQN and IW seems to be a good mechanism for decision making and opens the door to the incorporation of other AI

techniques that have proven to be effective in the Atari domain as CGP, from the case study about genetic programming [18].

In terms of runtime, the new agent has proven to be as fast as *SearchAgent* because the time the network spends to determine an action is insignificant compared to the time IW takes to perform its simulations.

A disadvantage of this agent is that a universal parameter configuration cannot be established for all video games, so it would be necessary to experiment in each of them to determine the ideal values of parameters such as *ratio* or *limite_simulacion*.

Another disadvantage is the networks training, which can be improved if we compare it with the results of *DeepMind* in its article about DQN [16]. However, the results of the experimentation show that less intelligent networks can be of help to IW to resolve games in certain video games. Without restrictions that limit the networks learning, the training could improve, which would have a positive effect on the results of *CombinationAgent*.

2.8 Socio-economic context

The solution has been proposed and implemented in the field of Atari 2600 games. However, the idea is adaptable and applicable in other areas. It could be extended to other video games and it would serve to build Non-Player Characters (NPC), that is, characters that execute tasks within the game themselves: attack, defend, move, etc.

Another area that could benefit is robotics. The developed solution could be used for a robot to solve tasks (displacements, manipulation of objects, etc.) reserved to date to people.

The impact of the solution would depend on the scope of application. In the field of robotics, it could help companies give more prominence to robots to solve certain tasks. This would free people from heavy or complex tasks, improving their quality of life. It could also end traditional jobs but, at the same time, could encourage new ones related to the construction and maintenance of robots.

In the entertainment industry, the solution could help develop more challenging and entertaining games. The positive consequences would be the increase of the industry's income, the increase of competitiveness and the creation of new jobs. The main negative consequence would be the addiction to video games, especially among the young population.

2.9 Regulatory framework

The developed work does not use personal information, does not compromise the safety of people or organizations and does not entail professional responsibilities. No law is applicable to the project.

The software on which the solution proposed in the project is based has free software licenses that offer freedom to modify, copy, publish, distribute and sell.

The software developed in this project also has the same freedoms. There is no patent for the idea, nor for its execution.

3 Introducción

En el presente documento se expone el Trabajo de Fin de Grado desarrollado. Se trata de un proyecto de tipo experimental en el que intervienen distintas ramas de la Inteligencia Artificial (planificación, aprendizaje por refuerzo, *Deep Learning*, redes de neuronas, etc.) para dar solución a cuestiones relacionadas con la industria del entretenimiento, en concreto, con el mundo de los videojuegos.

El proyecto se basa en investigaciones reconocidas en el ámbito de la Inteligencia Artificial, aplica conceptos de otros sectores de la informática como la Ingeniería del *Software* e incluye una exhaustiva experimentación que suma más de 300 horas de ejecución.

En esta introducción, se hablará resumidamente del contexto y los objetivos del proyecto, y de la motivación para desarrollar el tema elegido. También se explicará brevemente la estructura del documento.

3.1 Contexto y objetivos

La consola *Atari 2600* dispone de una gran cantidad de juegos que, en la actualidad, pueden ser ejecutados por un emulador multiplataforma llamado *Stella*. Tomando como base dicho emulador, diversos investigadores han desarrollado un *software* denominado *Arcade Learning Environment* (ALE) que abre las puertas al estudio de técnicas de Inteligencia Artificial sobre los juegos de esa consola. El objetivo de los investigadores es construir agentes que puedan resolver las partidas exhibiendo un comportamiento inteligente, es decir, eligiendo sabiamente la mejor acción en cada estado de la partida.

Actualmente, existen tres enfoques principales para resolver los juegos de Atari:

- ***Planning*.** Se trata de un grupo de algoritmos que, dado un estado de la partida, realizan simulaciones del futuro para averiguar qué acciones son mejores. IW es uno de los algoritmos que ofrece mejores resultados dentro de este grupo.
- ***Reinforcement learning*.** Son técnicas de aprendizaje por refuerzo que someten al agente a una sucesión de episodios de entrenamiento. De esta manera, el agente adquiere experiencia sobre cómo abordar las partidas para obtener la máxima puntuación. DQN es uno de los algoritmos más reconocidos de este enfoque.
- ***Genetic programming***. Es un método aplicado recientemente al dominio de Atari. Consiste en evolucionar programas de computador por medio de mecanismos propios de la evolución biológica para que resuelvan las partidas de manera inteligente.

Los enfoques son completamente diferentes en su planteamiento, tienen sus ventajas e inconvenientes, y no son perfectos. El **objetivo** fundamental de este trabajo es

desarrollar un nuevo agente basado en técnicas de Inteligencia Artificial que pueda tomar decisiones en tiempo real en entornos dinámicos como los videojuegos de Atari.

Para guiar el proyecto, se han marcado unos subobjetivos a cumplir:

1. Estudio del estado actual del tema del proyecto, incluyendo base teórica, técnicas de Inteligencia Artificial desarrolladas hasta el momento y herramientas *software* utilizadas por los investigadores.
2. Construcción de un nuevo agente que combine diferentes técnicas empleadas en el ámbito de los juegos de Atari (*reinforcement learning* y *planning*), siguiendo una metodología de desarrollo *software* que asegure el correcto análisis, diseño e implementación del agente.
3. Experimentación para poner a prueba el nuevo agente en múltiples videojuegos de Atari. Será exhaustiva, pero adaptada a las limitaciones propias del computador portátil que se usará. Su rendimiento se ve superado enormemente por las máquinas utilizadas en el ámbito de los videojuegos de Atari. Por ello, se establecerán de manera clara y justificada las limitaciones de los experimentos.
4. Análisis de los resultados obtenidos en la experimentación para averiguar cómo se comporta el nuevo agente, en comparación con los otros agentes estudiados en el estado del arte.

3.2 Motivación

Como se ha demostrado en numerosas ocasiones a lo largo del grado, una manera tremendamente efectiva de aprender es poner en práctica los conceptos vistos en clase en un entorno atractivo, que estimule el aprendizaje. Y no se me ocurre mejor entorno que el mundo de los videojuegos. Son diversas las asignaturas que han aprovechado la amenidad de videojuegos como *Pacman*, *Starcraft* o *Buscaminas* para fomentar el aprendizaje y la investigación.

Otro factor que ha influido en la decisión del tema del proyecto es el crecimiento espectacular de la industria del videojuego. Cada vez se invierte más en el desarrollo de métodos basados en Inteligencia Artificial para mejorar la credibilidad y calidad de los videojuegos. Además, muchas de las técnicas de Inteligencia Artificial estudiadas para la industria del entretenimiento pueden exportarse a otros campos de interés como el de la robótica (búsqueda de caminos, por ejemplo), lo cual es un aliciente más para investigar con videojuegos.

3.3 Estructura del documento

El documento consta de una serie de apartados que describen el desarrollo del trabajo:

- **Estado del arte.** En este punto se llevará cabo un profundo estudio del estado del tema escogido: conceptos teóricos esenciales, casos de estudio relevantes y

herramientas *software* existentes. En definitiva, se establecerá el contexto del tema de proyecto.

- **Planteamiento de la cuestión.** Basándose en el estado del arte, en este apartado se escogerá la cuestión a resolver durante el resto del trabajo, argumentando debidamente las razones de la elección.
- **Desarrollo de la cuestión.** En este punto se hablará de la construcción de un *software*, como paso intermedio para resolver la cuestión planteada en el apartado anterior. Se utilizará una metodología de desarrollo para guiar el análisis, diseño e implementación del *software*.
- **Evaluación.** Con el *software* implementado, se llevará a cabo una experimentación. Se describirá la metodología de evaluación utilizada, se ejecutarán los experimentos por etapas y se expondrán los resultados obtenidos (incluyendo gráficas y tablas).
- **Análisis de los resultados.** En este punto los resultados de la experimentación serán analizados.
- **Planificación del proyecto.** En este apartado se expondrá la organización del proyecto: fases en las que se ha dividido, tareas asignadas a cada fase, periodos (estimados y reales) de desarrollo de las diferentes tareas. Se incluirán gráficos de *Gantt* en los que se visualice la planificación del proyecto.
- **Entorno socioeconómico.** En este apartado se incluirá el presupuesto del proyecto. También se hablará del posible impacto del proyecto en diversos ámbitos: económico, social, ético.
- **Marco regulador.** En este punto se hablará sobre los aspectos legales relacionados con el trabajo: legislación vigente, licencias *software*, propiedad intelectual, etc.
- **Conclusiones.** Se expondrán las conclusiones fundamentales del proyecto, así como las opiniones personales, dificultades encontradas y posibles trabajos futuros.

Además, el documento incluye apartados complementarios como resumen, bibliografía, glosario de términos (siglas y definiciones), y apéndices.

4 Estado del arte

4.1 Base teórica

4.1.1 Inteligencia Artificial y la industria del entretenimiento

La Inteligencia Artificial (IA) es una rama de las Ciencias de la Computación dedicada a desarrollar técnicas que permiten a los computadores resolver problemas que requieren de las capacidades de aprendizaje o razonamiento propias de los seres humanos.

Computing Machinery and Intelligence de Alan Turing [22] podría considerarse el punto de partida para la Inteligencia Artificial. En el artículo, Turing propone el "juego de la imitación", una prueba que tiene como objetivo determinar si una máquina es capaz de manifestar un comportamiento inteligente parecido al que tendría una persona.

Turing pronosticó que las máquinas superarían la prueba antes de que finalizase el siglo XX. Aunque esta predicción no se cumpliese, los grandes avances de las últimas décadas son innegables. Hoy en día, tareas como el procesamiento del lenguaje natural, el procesamiento de imágenes, el reconocimiento de patrones, etc. pueden ser parcialmente delegadas a las máquinas.

Las empresas han reconocido la gran utilidad de la Inteligencia Artificial para mejorar sus productos o incrementar la eficiencia y eficacia de su modelo de negocio. Son muchos los sectores que hacen uso de la Inteligencia Artificial: atención al cliente, finanzas y banca, marketing, medicina y robótica, entre otros muchos.

Un área que está aprovechando especialmente los avances en IA es la industria del entretenimiento, que engloba al deporte, videojuegos, cine, radio, televisión, danza y música, principalmente.

La de los videojuegos es quizás la industria más potente dentro del entretenimiento. Según un estudio de 2017 de la *Entertainment Software Association* (ESA) [23], solo en Estados Unidos se vendieron 24.5 billones de dólares en videojuegos durante 2016. En aquel momento, EE. UU. contaba con 65.000 empleos directos, y con un total de 220.000 empleos relacionados con la industria del videojuego.

Este sector también destaca por haber incorporado en los últimos años una gran cantidad de técnicas de IA con el objetivo de mejorar la calidad de los videojuegos: que sean más dinámicos e imaginativos, que no cansen rápidamente al jugador y supongan todo un desafío para ellos.

Las principales líneas de acción son la búsqueda de caminos, la toma de decisiones inteligentes y la generación automática de contenido.

- **Búsqueda de caminos.** Se trata de una parte importante en videojuegos que disponen de mapas y por los cuales los personajes pueden moverse de un punto a otro sorteando obstáculos. En este tipo de búsqueda, no interesa tanto la

optimalidad como la naturalidad del camino, es decir, que la trayectoria seguida por el personaje no sea demasiado angulosa. Existe una gran variedad de algoritmos de búsqueda que se adaptan al tipo de mapa (continuo/discreto, niebla de guerra) y de juego (búsqueda en tiempo real, memoria disponible limitada). Un ejemplo destacado de algoritmo para la búsqueda de caminos es A* [24].

- **Generación automática de contenido.** Delegar en un computador la tarea de generar historias, paisajes y niveles de un videojuego puede aportar un punto de creatividad no alcanzable por la imaginación humana. Algunos técnicas: (1) la planificación automática, que ha demostrado ser útil para desarrollar narrativa [25], y (2) los fractales, que son imprescindibles para construir terrenos [26] y objetos como árboles [27].

- **Toma de decisiones.** Para dar mayor realismo al juego, es necesario que los personajes se comporten de manera inteligente, por lo que las acciones que ejecuten en cada momento deben ser coherentes. Las redes jerárquicas de tareas (HTN) [28] o los árboles de comportamiento [29] son ejemplos de métodos para mejorar el proceso de toma de decisiones.

4.1.2 Aprendizaje Automático

El Aprendizaje Automático (también conocido como *Machine Learning* o ML) [1] es un subcampo de la Inteligencia Artificial que tiene como objetivo definir algoritmos que permitan a un sistema *software* aprender a resolver tareas específicas. Para abordar apropiadamente la tarea, el sistema debe ser entrenado suministrándole un conjunto de ejemplos de entrenamiento.

Las técnicas de ML pueden clasificarse según el tipo de aprendizaje que llevan a cabo. Los tres tipos de aprendizaje principales son:

- **Aprendizaje supervisado:** Los ejemplos de entrenamiento suministrados al sistema durante el aprendizaje son previamente etiquetados. El objetivo del aprendizaje es que el sistema prediga la etiqueta asociada a cualquier entrada. Las tareas que pueden resolverse por medio del aprendizaje supervisado son clasificación y regresión.

- **Aprendizaje no supervisado:** Los ejemplos de entrenamiento suministrados al sistema durante el aprendizaje no están etiquetados. Una de las tareas más comunes dentro de este tipo de aprendizaje son las de *clustering*, en las que el sistema debe agrupar en subconjuntos los ejemplos que guarden cierto parecido.

- **Aprendizaje por refuerzo (*Reinforcement Learning*):** Este modelo de aprendizaje se basa en la teoría psicológica del *conductismo*. El objetivo es que

un agente ubicado en un entorno sea capaz de decidir qué acciones tomar en cada estado para maximizar un refuerzo acumulado.

4.1.3 Aprendizaje por Refuerzo

El aprendizaje por refuerzo (también conocido como *Reinforcement Learning*, o RL) [2] se utiliza para resolver tareas con las siguientes características:

1. Hay dos elementos: un agente y un entorno.
2. El agente dispone de un conjunto de acciones con el que interacciona con el entorno.
3. La interacción agente-entorno se desarrolla en tiempo discreto. Como puede observarse en la Ilustración 1, el agente observa el estado actual del entorno y ejecuta una acción en cada instante de tiempo. La acción provoca que el entorno cambie de estado y devuelva al agente una recompensa (refuerzo) que puede ser negativa, positiva o neutra.

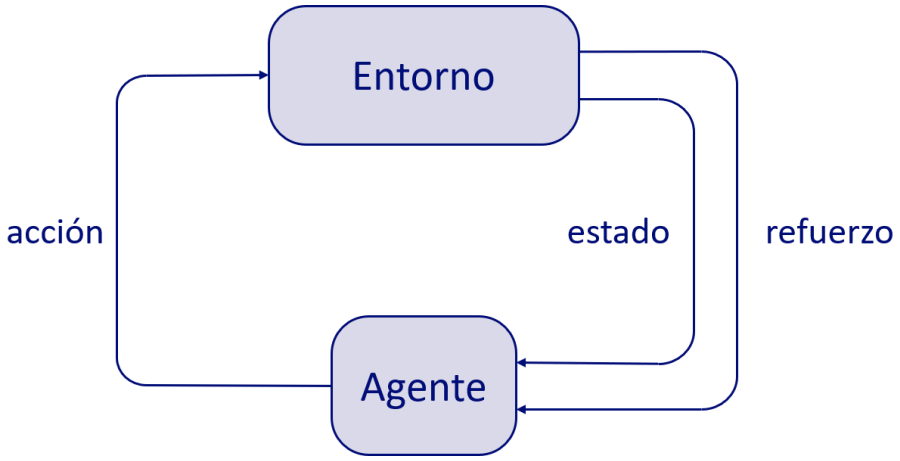


Ilustración 1. Esquema de interacción agente-entorno en un instante de tiempo

4. El objetivo del agente es alcanzar un estado final partiendo de un estado inicial del entorno, aplicando una secuencia de acciones que maximicen el refuerzo total.

Para conseguir que el agente se comporte de manera inteligente dentro del entorno, se necesita un entrenamiento. Se trata de un proceso iterativo de prueba y error que utiliza los refuerzos otorgados por el entorno para que el agente aprenda a discernir qué acción es mejor en cada estado del entorno.

La tarea típica que puede resolverse con aprendizaje por refuerzo es el desarrollo de un agente capaz de resolver un videojuego con la mejor puntuación posible, sin intervención humana (*Non-player character*).

Uno de los algoritmos más destacados del aprendizaje por refuerzo es ***Q-Learning*** [30], cuyo objetivo es generar una política óptima para el agente por medio de un entrenamiento basado en una sucesión de episodios.

En la Tabla 1 se presenta la versión de *Q-Learning* con la estrategia *ϵ-greedy* [31]:

Inicializar tabla $Q(s,a), \forall\, s \in S, \forall\, a \in A$.
FOR $episodio\ in\ [0 \dots M]$:
    Inicializar el estado inicial s_0 de forma aleatoria.
    FOR $t\ in\ [0 \dots T]$:
        Con probabilidad ϵ se elige una acción a_t aleatoria.
        En caso contrario se elige la acción $a_t = \arg\max_a Q(s_t, a)$
        Ejecutar a_t y recibir el nuevo estado s_{t+1} y el refuerzo r_t.
        $Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a)\right)$

Tabla 1. Pseudocódigo de Q-Learning

Donde:

- $\boldsymbol{S}$ es el espacio de estados del entorno; $\boldsymbol{A}$ es el conjunto de acciones del agente.
- $\boldsymbol{\alpha}$ es el *ratio de aprendizaje*, que determina cómo de rápido aprende a lo largo del entrenamiento. [$\alpha \in \mathbb{R} \mid 0 \leq \alpha \leq 1$]
- $\boldsymbol{\gamma}$ es el *factor de descuento*, que determina la importancia de los refuerzos a largo plazo. [$\gamma \in \mathbb{R} \mid 0 \leq \gamma \leq 1$]
- $\boldsymbol{\epsilon}$ de la estrategia de exploración-explotación *ϵ-greedy*, se refiere a la probabilidad de que en un instante t de tiempo el agente explore el entorno ejecutando una acción completamente aleatoria. Una estrategia de exploración-explotación es necesaria durante el entrenamiento con el fin de que el agente primero explore el espacio de estados y después explote los estados más prometedores. Por ello, el valor de $\boldsymbol{\epsilon}$ suele disminuir con el avance del entrenamiento. [$\epsilon \in \mathbb{R} \mid 0 \leq \epsilon \leq 1$]

Cuando el entrenamiento ha finalizado, el agente dispone de una tabla Q formada por todos los valores $Q(s,a)$. Estos valores representan lo buena que es la acción a dado el estado s. Por tanto, dado un estado s, la acción a' elegida por el agente es aquella con el mayor valor $Q(s,a)$.

$$a' = \arg\max_a Q(s,a)$$

4.1.4 Redes de Neuronas Artificiales

Las Redes de Neuronas Artificiales [3] [32] son modelos computacionales bioinspirados que forman parte de las técnicas de Aprendizaje Automático y, por consiguiente, de las técnicas de Inteligencia Artificial. Se utilizan para resolver tareas de aprendizaje supervisado (clasificación, regresión) y no supervisado (*clustering*).

Estos modelos matemáticos se basan en las redes de neuronas animales. Aun así, lo único que tienen vagamente en común es la topología de red que forman las neuronas al interconectarse, y la capacidad de las neuronas de transmitir información (impulsos) y de activarse o no dependiendo de las activaciones de las neuronas conectadas.

Existen diversos tipos de redes de neuronas (Perceptrón Multicapa, Adaline, Redes de Base Radial, Redes Convolucionales, etc.), pero en general pueden describirse de la siguiente manera:

- La unidad fundamental es la neurona (véase la Ilustración 2). Dispone de un conjunto de n entradas y un valor de umbral U. Cada entrada tiene asociado un peso W_j, que determina el grado de importancia de la entrada. Las señales de entrada X_j son ponderadas por sus respectivos pesos W_j y posteriormente sumadas junto al umbral U para obtener el sumatorio $\sum$. Sobre el valor resultante del sumatorio se aplica la función de activación $F(\sum)$, que genera una única salida S, conocida como "activación" de la neurona.

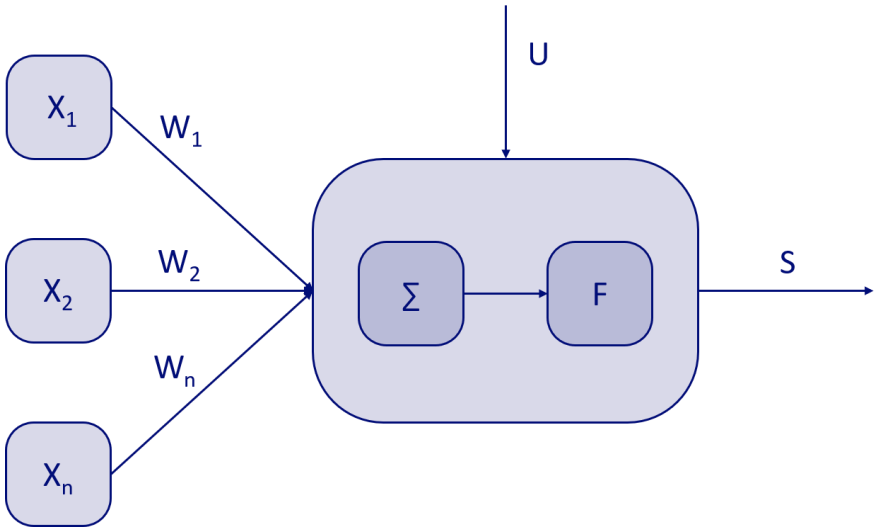


Ilustración 2. Esquema de una neurona

- Las neuronas están interconectadas, siguiendo normalmente una organización por capas (véase la Ilustración 3). Se distingue una capa de entrada, constituida por neuronas que reciben la entrada global de la red, y una capa de salida, formada por neuronas que generan la salida global de la red. También pueden existir capas intermedias, denominadas capas ocultas. Las activaciones de las

neuronas de una capa constituyen las entradas de las neuronas de la capa siguiente. La activación de una neurona depende de las activaciones de las neuronas conectadas a ella.

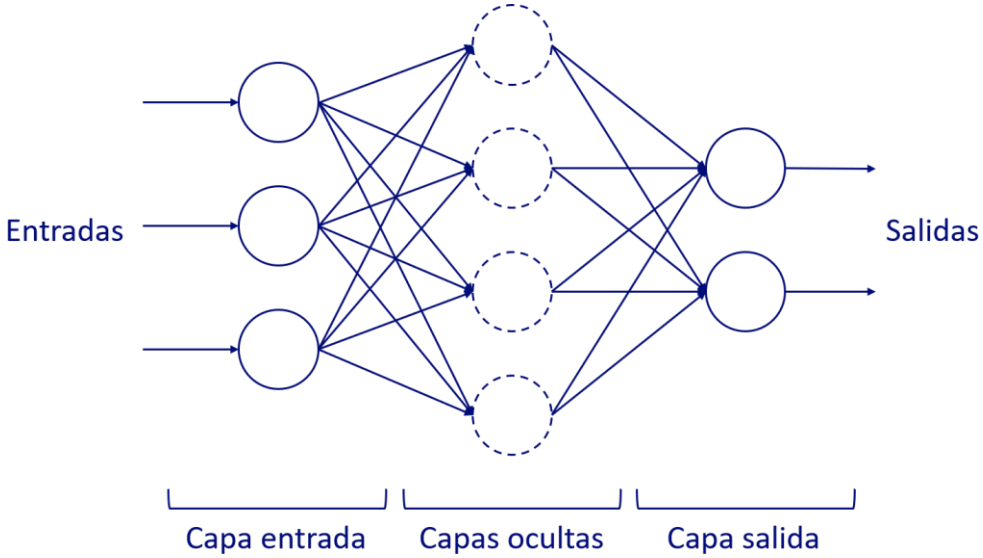


Ilustración 3. Esquema de una red de neuronas genérica

Las redes de neuronas aprenden a resolver problemas por medio de un proceso de aprendizaje que necesita de un conjunto representativo de ejemplos (también llamados patrones) de entrenamiento. El proceso de entrenamiento consiste en presentar como entrada de la red cada uno de los ejemplos del conjunto.

El aprendizaje radica en ajustar los pesos de la red, es decir, variar sus valores. Estas variaciones se llevan a cabo tras presentar cada ejemplo de entrenamiento.

El aprendizaje concluye cuando se cumple algún criterio de parada (por ejemplo, que los pesos no sufran modificaciones). El entrenamiento ha sido efectivo si la red es capaz de generalizar, es decir, aplicar el conocimiento aprendido para ejemplos distintos a los de entrenamiento (no presentados a la red previamente).

4.1.5 *Deep Learning*

Aprendizaje Profundo (más conocido como *Deep Learning* o DL) [33] [4] [34] se refiere a un grupo de técnicas de Aprendizaje Automático que utilizan redes de neuronas artificiales con múltiples capas. El término *redes de neuronas profundas* es equivalente a *Deep Learning*.

Las redes de neuronas profundas cuentan con una jerarquía de niveles. Cada nivel de la jerarquía actúa sobre un conjunto de características del problema, basado en el nivel anterior. Los niveles sucesivos son capaces de aprender conceptos de mayor complejidad y abstracción, ya que agregan características de los niveles previos.

Las técnicas de DL se diferencian de las técnicas de ML tradicionales en varios aspectos:

- **Gran cantidad de datos.** Los algoritmos de *Deep Learning* son apropiados cuando el conjunto datos es muy grande y presenta una elevada dimensionalidad. Las redes de neuronas profundas necesitan altos volúmenes de datos para obtener resultados aceptables.
- **Ingeniería de características.** Se refiere a la preparación del conjunto de datos que se utilizará para entrenar un sistema con un algoritmo de ML. Mientras que los métodos tradicionales necesitan que una persona seleccione las características manualmente para garantizar el éxito del aprendizaje, las técnicas de DL son capaces de construir el conjunto de características sin necesidad de supervisión.

Las aplicaciones potenciales de las redes de neuronas profundas son muy diversas: reconocimiento de imágenes, reconocimiento de voz, clasificación de vídeo, procesamiento del lenguaje natural, etc.

4.1.6 Redes neuronales convolucionales

Una red neuronal convolucional (también conocido como *Convolutional Neural Network* o CNN) [5] [34] es un tipo de red de neuronas artificiales especializado en resolver tareas de visión artificial, como la clasificación de imágenes.

A diferencia de las redes de neuronas estándar, las CNN no necesitan como paso previo la extracción de características de las imágenes para construir un conjunto de datos de entrenamiento. La arquitectura de las CNN está pensada para recibir como entrada imágenes en bruto, y obtener por sí misma las características relevantes de la imagen.

Las CNN siguen una arquitectura basada en capas. Pueden distinguir 4 componentes:

- **Capa de entrada.** Recibe como entrada la imagen en bruto.
- **Capa de convolución.** Se aplican filtros a las entradas de la capa. Un *filtro* (véase la Ilustración 4) es una matriz de pesos w_{ij} que se aplica sobre una región de la entrada (denominada *campo receptivo* p) siguiendo la expresión:

$$y = \sigma\left(\sum_{i,j} w_{ij} \cdot p_{ij}\right)$$

  Donde σ es la función de activación propia de la red, comúnmente la función ReLU.

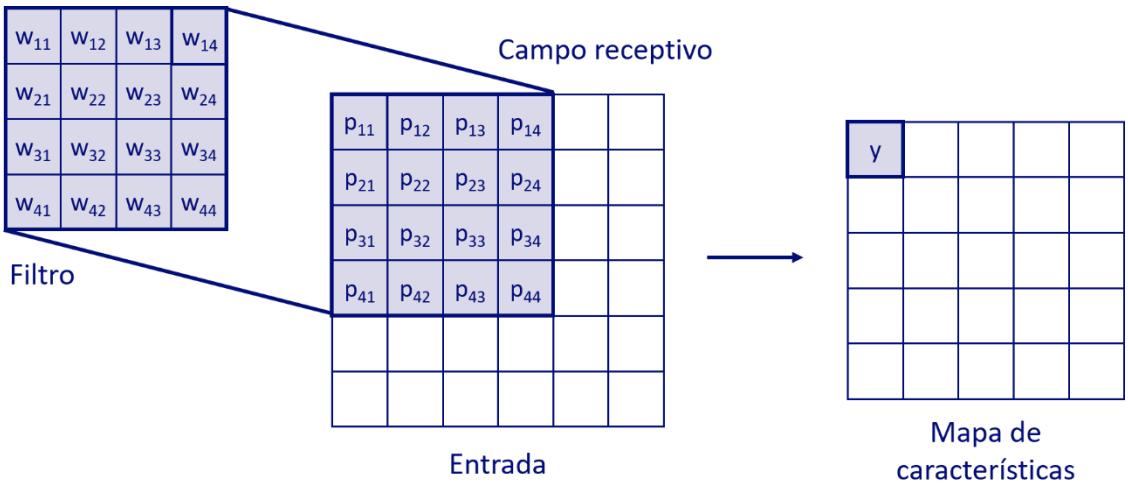


Ilustración 4. Aplicación de filtro 4x4 sobre una entrada

El filtro se desplaza por toda entrada, y todas las salidas y que se generan dan lugar a una matriz denominada *mapa de características*. Las entradas de una capa de convolución pueden ser imágenes si la capa anterior es la capa de entrada, o mapas de características si la capa anterior no es la capa de entrada.

El hecho de que utilicen un mismo filtro para toda la entrada reduce la cantidad de pesos de la red. Además, es posible que una capa de convolución utilice más de un filtro para la entrada. De esta forma, la red puede detectar distintas características de la entrada.

- **Capa de *sub-sampling*.** Se encarga de disminuir el tamaño de la entrada. Para ello, realiza una transformación de los campos receptivos p_i para obtener un valor de salida y_i. La transformación aplicada puede ser el máximo valor del campo perceptivo, la media de los valores, etc.

  En la Ilustración 5 se muestra la reducción de la entrada:

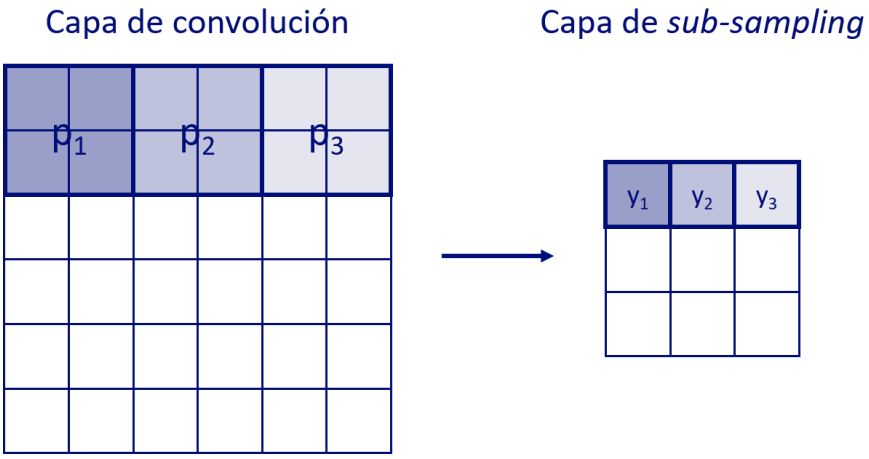


Ilustración 5. Reducción del tamaño de entrada 6x6 a dimensión 3x3

Normalmente, después de una capa de convolución hay una capa de *sub-sampling*.

- **Red estándar.** Habitualmente es un perceptrón multicapa (MLP). Se sitúa después de la sucesión de capas de convolución y *sub-sampling*. La salida de la última capa de *sub-sampling* es la entrada de la red estándar. Esta entrada es el resultado de la extracción de características relevantes de la imagen por parte de las capas de convolución y *sub-sampling*.

En la Ilustración 6 se presenta un esquema de la arquitectura general de una CNN:

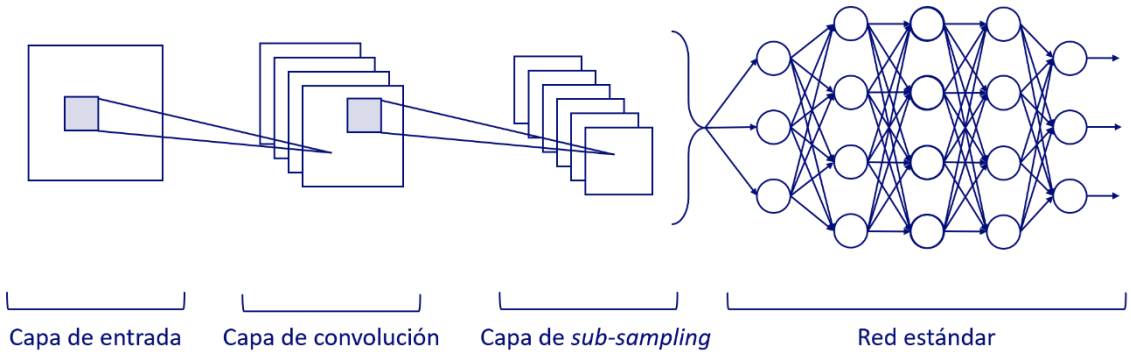


Ilustración 6. Arquitectura CNN (1 capa de convolución; 1 capa de sub-sampling)

4.1.7 Búsqueda

Existe un conjunto de problemas que, a falta de un procedimiento óptimo, solo pueden ser resueltos por medio de un proceso de búsqueda [6]. Estos problemas pueden definirse como una tupla $< S, A, s_o, G >$, donde:

- $\boldsymbol{S}$ es el espacio de estados del problema. Contiene todos los posibles estados alcanzables.
- $\boldsymbol{A}$ es el conjunto de acciones que pueden ejecutarse sobre un estado del problema.
- $\boldsymbol{s_o}$ se refiere al estado inicial, a partir del cual se desarrolla la búsqueda.
- $\boldsymbol{G}$ se refiere al conjunto de estados meta.

El objetivo de la búsqueda es, dado un estado inicial s_o y un conjunto de acciones A, alcanzar un estado meta $s_f \in G$ aplicando una secuencia de acciones $a \in A$ desde el estado inicial s_o. Los algoritmos de búsqueda se dedican a llevar a cabo esta tarea de búsqueda.

Existen numerosos algoritmos, que podrían categorizarse en dos bloques, según el grado de conocimiento que se tenga sobre el problema:

- **Búsqueda no informada (ciega).** No se tiene información sobre características del problema que permitan optimizar el proceso de búsqueda. Estos algoritmos

tienden a recorrer todo el espacio de estados de forma sistemática, sin ningún criterio heurístico. Los algoritmos de búsqueda en amplitud y profundidad [6] son no informados.

- **Búsqueda heurística.** Se aplican estrategias heurísticas para reducir el cómputo del algoritmo. Estas heurísticas son un conocimiento parcial sobre la proximidad de un estado a la solución. Algoritmos como A* [24] o IDA* [35] son de búsqueda heurística.

Los algoritmos de búsqueda también pueden clasificarse basándose en estos tres criterios:

1. **Completitud.** El algoritmo encuentra la solución al problema, si existe.
2. **Optimalidad.** El algoritmo encuentra la solución óptima (de menor coste).
3. **Eficiencia.** Relacionado con la medida de complejidad del algoritmo de búsqueda, es decir, cantidad de recursos temporales o espaciales necesarios por el algoritmo para hallar la solución al problema.

4.1.8 Computación evolutiva

La computación evolutiva [7] comprende aquellas técnicas de Inteligencia Artificial capaces de resolver problemas emulando los principios de la evolución biológica. Resulta especialmente útil para problemas en los que hallar la solución óptima no es posible.

Las técnicas de computación evolutiva utilizan el concepto de *población* de individuos. Cada individuo es una solución codificada del problema y es evaluada según una función de adecuación o *fitness*, que determina lo buena que es la solución.

El objetivo es evolucionar la población en sucesivas iteraciones (o generaciones) para obtener individuos cada vez más aptos, es decir, mejores soluciones del problema. En general, el procedimiento seguido por los algoritmos de computación evolutiva es el siguiente:

1. **Inicialización.** Se genera la población inicial. Los individuos suelen obtenerse de forma aleatoria o guiada según algún criterio.

2. **Selección.** Simula el proceso de selección natural Darwiniana, que establece que los individuos más aptos tienen mayor probabilidad de sobrevivir, reproducirse y trasmitir sus genes a la descendencia. En computación evolutiva los individuos son evaluados con la función de *fitness*, y un operador de *selección* se encarga de elegir los individuos que se reproducirán, que con mayor probabilidad serán los de mayor valor de *fitness*.

3. **Sobrecruzamiento.** Los individuos seleccionados intercambian sus genes entre ellos por medio de un operador de *sobrecruzamiento*. Este operador especifica qué genes se intercambian entre cada par de individuos.

4. **Inserción y reemplazo.** Se construye una nueva población basándose en la población actual y en los individuos resultantes del sobrecruzamiento. Existen múltiples estrategias de inserción y reemplazo.
5. **Mutación.** Los individuos de la nueva población sufren mutaciones en sus genes por medio de un operador de *mutación*. Esto permite conservar la variabilidad genética de la población. El operador de mutación más básico es la modificación de genes aleatorios de cada individuo.
6. **Parada.** Con cada generación, se comprueba un criterio de finalización. Hay mucha variedad: se alcanzan n generaciones, se supera el tiempo máximo establecido, no se encuentra un mejor individuo después de n generaciones, etc. Si se cumple el criterio, el algoritmo termina. En caso contrario, vuelve al paso 2.

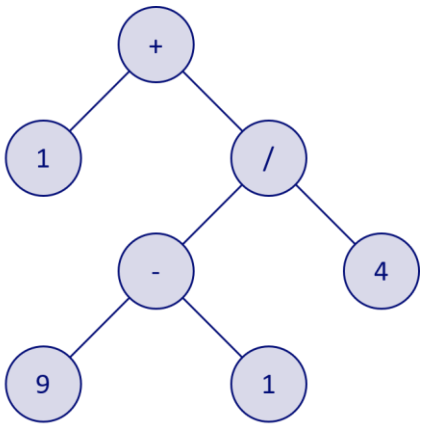
Aunque este es el procedimiento genérico, las técnicas de computación evolutiva son muy flexibles y los operadores de selección, recombinación, inserción, reemplazo y mutación pueden personalizarse para mejorar su desempeño.

El éxito de la computación evolutiva depende principalmente de tres factores: (1) la calidad de la codificación de los individuos, (2) la calidad de la función de *fitness* y (3) los operadores genéticos elegidos.

4.1.9 Programación genética

La programación genética [7] es una técnica de computación evolutiva especializada en resolver problemas cuya solución sea un programa de computador. Los individuos de la población se codifican como *expresiones-S* (véase la Ilustración 7), que son árboles en los que los nodos intermedios son símbolos *operadores* (o *funciones*) y los nodos hoja son símbolos *operandos*.

Símbolos	
Operadores	Operandos
+, -, /, *	1, 2, 3, 4, 5, 6, 7, 8, 9



(1+((9-1)/4))

Ilustración 7. Especificación de símbolos del problema (izquierda) y codificación de individuo (derecha)

El procedimiento genérico seguido por un algoritmo de programación genética se expone en la Tabla 2:

```
Inicialización de la población inicial.
WHILE criterio de parada no cumplido:
        Evaluación de cada individuo con la función de fitness.
        Creación de población intermedia vacía.
        Inserción de individuos más aptos de la población en población intermedia.
        WHILE población intermedia no llena:
                Elección entre operador de mutación o sobrecruzamiento.
                IF sobrecruzamiento elegido:
                        Selección de dos individuos de la población.
                        Sobrecruzamiento de los individuos.
                ELSE IF mutación elegida:
                        Selección de un individuo de la población.
                        Mutación del individuo.
                Inserción de los nuevos individuos en la población intermedia.
        Nueva población ← Población intermedia.
```

Tabla 2. Procedimiento de programación genética

Los operadores de mutación principales son:

- **Mutación terminal simple.** Se escoge de forma aleatoria un nodo hoja y se sustituye el símbolo *operando* por otro.
- **Mutación funcional simple.** Se escoge de forma aleatorio un nodo intermedio y se sustituye el símbolo *operador* por otro.
- **Mutación de subárbol.** Se escoge de forma aleatoria un subárbol y se sustituye por uno nuevo aleatorio.

El operador de sobrecruzamiento consiste en elegir de forma aleatoria un subárbol en cada uno de los dos individuos e intercambiarlos.

4.1.10 Atari

Atari [8] [9] es una productora de videojuegos estadounidense fundada en 1972. Es considerada como la iniciadora de la industria del videojuego gracias a *Pong*, un juego basado en el tenis de mesa que se hizo enormemente popular en las salas recreativas.

El éxito que supuso *Pong* permitió a Atari expandirse en el joven mercado de los videojuegos. La empresa comenzó a desarrollar sus propias videoconsolas comenzando en 1975 con la *Atari Pong* [36], una consola doméstica que incluía *Pong* como único videojuego.

La consola registró unas ventas de aproximadamente 55.000 unidades, casi nada si lo comparamos con los 30 millones de unidades que vendería Atari años más tarde con su siguiente videoconsola doméstica, la *Atari 2600* [37].

La *Atari 2600*, lanzada en 1977 en EE. UU., popularizó el concepto de cartuchos intercambiables, que sentó la base para las futuras generaciones de videoconsolas. Gracias a los cartuchos extraíbles la consola llegó a tener un amplio catálogo de videojuegos, a diferencia de su predecesora.

La consola contaba con las siguientes especificaciones [38]:

- Procesador de 8 bits *MOS Technology 6507*.
- Memoria RAM de 128 Bytes.
- Procesador de video que permitía visualizar imágenes de 160 píxeles de ancho y 210 píxeles de alto, con una gama de 128 colores.
- **Entrada.**
  - Dos puertos para conectar periféricos (*joysticks*, *trackballs, volantes*, etc.).
  - Interruptores de encendido y apagado, señal de TV, nivel de dificultad, *Select* y *Reset*.
  - Un puerto para cartuchos de videojuegos.
- **Salida.** Señal de vídeo y audio para televisión.

Los cartuchos de la *Atari 2600* disponían de una memoria ROM de 4KB utilizada para almacenar un único videojuego. Para algunos juegos, el cartucho podía contener más de 32 KB de memoria ROM y 256 Bytes adicionales de memoria RAM.

Los juegos de la *Atari 2600* se caracterizaban por presentar una gama muy reducida de colores, texturas con bajo nivel de detalle y una mecánica de juego sencilla. Dentro de su extenso catálogo, pueden destacarse los siguientes títulos: *Pong*, *Breakout, Asteroids, Defender, Montezuma, Pac-Man*, *Mario Bros*, *Donkey Kong*, *Space Invaders*, etc.

En la Ilustración 8 se muestran las capturas de algunos juegos de la consola:

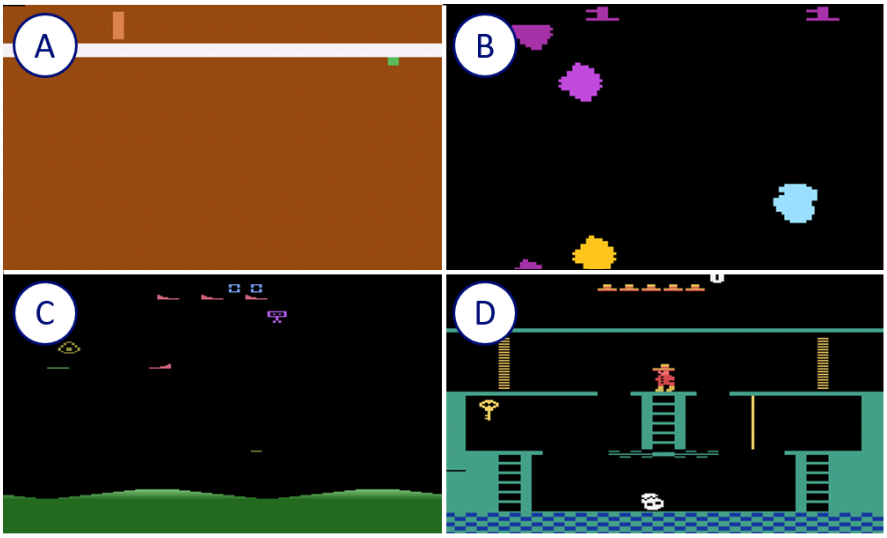


Ilustración 8. Capturas de algunos juegos de Atari 2600: Pong (A), Asteroids (B), Star Gunner (C) y Montezuma (D) (obtenidas de Atarimania [39] [40] [41] [42])

Atari trató de mantenerse al frente del sector de las videoconsolas durante las décadas de los 80 y 90 con nuevos lanzamientos: *Atari 5200*, *Atari 7800*, *Atari XEGS*, *Atari Lynx* y *Atari Jaguar*. Sin embargo, con ninguna de ellas consiguió repetir el éxito de la *Atari 2600*.

4.2 Herramientas *software*

4.2.1 *Arcade Learning Environment* (ALE)

ALE es *software* presentado por Marc G. Bellemare y su equipo en su artículo *The Arcade Learning Environment: An Evaluation Platform for General Agents* [15]. Se trata de un *framework* que permite a investigadores desarrollar agentes inteligentes sobre más de 50 videojuegos de Atari, utilizando técnicas de Inteligencia Artificial como aprendizaje por refuerzo o búsqueda/planificación.

ALE está construido sobre *Stella*, un emulador de la consola *Atari 2600*. *Stella* sirve al desarrollador como interfaz para interactuar con el entorno, de forma que no tenga que preocuparse por los detalles de emulación.

La interacción entre el agente y el entorno se produce en el marco de un episodio. Un episodio es, en el ámbito de ALE, una partida de un videojuego. La interacción dentro de un episodio se representa gráficamente en la Ilustración 9, y es la siguiente:

1. El agente recibe una observación inicial del episodio, que es información sobre el estado de la partida. Pueden ser 128 Bytes de memoria RAM de la consola emulada, o una matriz de 120 píxeles de ancho y 210 píxeles de alto correspondiente a la señal de vídeo en el *frame* actual de la partida.

2. El agente elige una acción, y se la transmite al emulador. El emulador dispone de un conjunto de 18 acciones (las permitidas por el *joystick* de la consola). Sin embargo, cada juego tiene un conjunto de acciones mínimas permitidas.
3. El emulador ejecuta la acción y devuelve al agente una nueva observación (siguiente *frame*) y un refuerzo. El refuerzo se corresponde normalmente con la diferencia de puntos entre 2 *frames* consecutivos de una partida. Si el episodio no ha concluido, vuelta al paso 2.

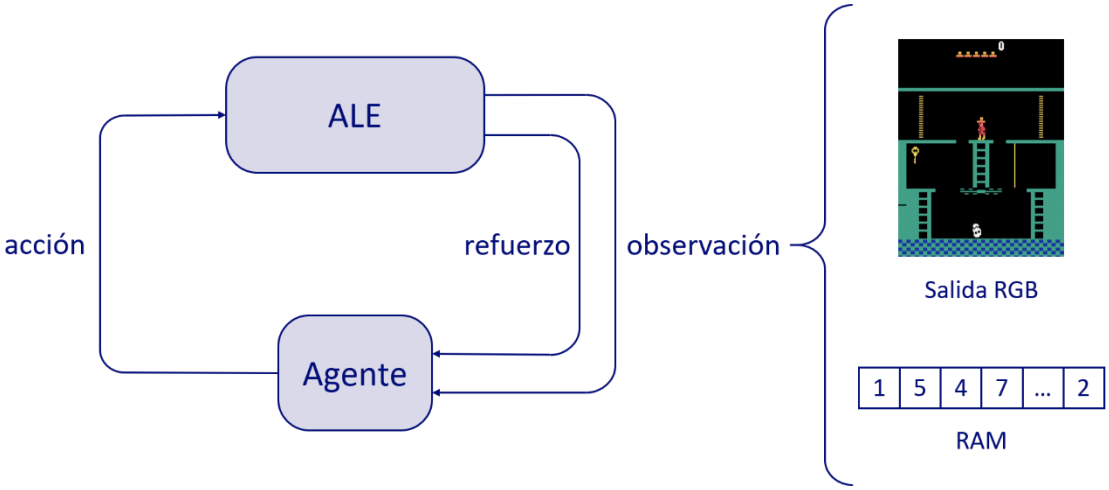


Ilustración 9. Esquema de relación entre agente-entorno en un episodio de Montezuma

Una característica interesante de ALE es que permite guardar el estado del emulador (memoria RAM, registros, contadores, etc.), y restaurarlo en cualquier momento. Esto resulta de gran utilidad en tareas de planificación de acciones, ya que permite al agente decidir su siguiente acción basándose en predicciones de la partida, sin perder el estado actual.

El código de ALE se encuentra disponible en un repositorio de *GitHub* [10].

4.2.2 *Gym*

Gym [11] es un *framework* de desarrollo y evaluación de técnicas de aprendizaje por refuerzo. Se trata de una librería que reúne una colección de entornos sobre los que pueden evaluarse los algoritmos. *Gym* está escrito en *Python*, por lo que es compatible con librerías de computación numérica *Tensorflow*, *Theano* o *Pytorch*.

Los entornos disponibles son los siguientes:

- ***Classic Control.*** El entorno contiene problemas clásicos de control que suelen resolverse con aprendizaje por refuerzo, como *Cart Pole*[1].

[1] Problema *Cart Pole*: https://www.youtube.com/watch?v=Lt-KLtkDlh8

- ***Algorithmic*.** Este entorno tiene como objetivo aprender algoritmos a partir de ejemplos. Los algoritmos deben resolver problemas sencillos como inversión del orden de listas, copia de cadenas, suma de números, etc.
- ***Atari*.** Integra el *framework Arcade Learning Environment* [15]. Debido a esto el entorno puede emular un gran número de juegos de la consola *Atari 2600* y facilita el desarrollo de agentes que aprendan a ganar partidas con la máxima puntuación posible.
- ***MuJoCo* (*Multi-Joint dynamics with Contact*).** Este entorno provee un conjunto de tareas de control sobre un potente simulador de físicas, utilizado para el desarrollo de robots, gráficas y animaciones.
- ***Box2D*.** Al igual que *MuJoCo*, ofrece un conjunto de tareas de control a resolver. La diferencia es que se utiliza un simulador de físicas en 2 dimensiones, denominado *Box2D*.
- ***Robotics*.** Simula robots en un entorno tridimensional. El objetivo es entrenarles para abordar tareas concretas, como la manipulación de objetos.
- ***Toy text*.** Entornos sencillos a modo de iniciación.

El aprendizaje en *Gym* se lleva a cabo en "episodios" en los que un agente trata de resolver una tarea dentro del entorno. La interacción entre el entorno y el agente sigue el esquema de la *Ilustración 1.* Se realiza por medio de una función denominada *step*, que indica al entorno la siguiente acción que el agente desea ejecutar. Las acciones disponibles están definidas para cada entorno. Basándose en la acción escogida, el entorno actualiza su estado y devuelve al agente la siguiente información:

- ***Observation*.** Información sobre el estado del episodio, que es observable por el agente. Le sirve para elegir la siguiente acción. Esta información depende del entorno (por ejemplo, en el entorno *Atari* sería una matriz de píxeles).
- ***Reward*.** Refuerzo obtenido al haber ejecutado la acción previa en el entorno.
- ***Done*.** Indica que la tarea (episodio) ha terminado, y que el entorno puede reiniciarse.
- ***Info*.** Información adicional utilizada con fines de depuración.

4.2.3 Librerías de computación numérica

Las librerías de computación [43] [44] numérica son aquellas que proporcionan herramientas para desarrollar sistemas que, entre otras cosas, aprendan a resolver tareas de aprendizaje automático y *Deep Learning*. Entre las muchas alternativas destacan las dos siguientes:

- ***TensorFlow*** [12]**.**

  Se trata de una librería de código libre desarrollada por *Google*. La computación numérica se representa por medio de grafos de flujo de datos estáticos. Los nodos del grafo equivalen a las operaciones matemáticas. Las conexiones entre

nodos del grafo representan los "tensores", que son *arrays* multidimensionales de datos.

La arquitectura de *TensorFlow* es lo suficientemente flexible como para permitir su ejecución en computadoras variadas: escritorio, servidores, móviles, etc. *TensorFlow* proporciona una API en lenguajes populares como *Python*, *C++* o *Java*. También es compatible con la aceleración GPU, lo que permite que tareas pesadas de *Deep Learning*, que procesan grandes cantidades de datos, puedan ver incrementado su rendimiento.

- ***PyTorch*** [13]**.**

  Se trata de una librería de código libre para el desarrollo de sistemas basados en *Deep Learning*. Se encuentra en fase *beta* y está muy integrado en *Python*, por lo que el desarrollo y depuración del código generalmente es fácil. Al igual que *TensorFlow*, *PyTorch* utiliza grafos de flujo de datos, pero en este caso dinámicos. También permite ejecutar sobre GPUs.

4.2.3.1 Comparativa

En la Tabla 3 se realiza una breve comparativa [43] [44] de las librerías de computación numérica estudiadas:

Características	*PyTorch*	*TensorFlow*
Uso	Para proyectos de pequeña escala o con fines de investigación.	Para proyectos con despliegue gran escala o entre plataformas.
Lenguajes soportados	*Python*	*Python*, *C/C++*, *Java*, *Go*
Instalación	Sencilla y rápida (vía *pip*[2])	Sencilla, aunque es necesario instalar distintos paquetes para ejecutar sobre GPUs.
Desarrollo y depuración	Fácil	Requiere experiencia
Documentación	Completa	Completa
Comunidad	Reducida, en crecimiento	Grande y competente

Tabla 3. Comparativa TensorFlow – PyTorch

[2] Sistema de gestión de paquetes que facilita la instalación de *software* para *Python*

4.3 Casos de estudio

4.3.1 *Playing Atari with Deep Reinforcement Learning*

Este artículo [14], publicado en 2013 por *DeepMind*, formula y evalúa un algoritmo que relaciona los conceptos *Deep Learning* y *Reinforcement Learning*: *Deep Q-Learning* (DQN).

La intención de DQN es entrenar agentes inteligentes para entornos cuyo espacio de estados es extremadamente grande. Para este tipo de entornos, los algoritmos tradicionales de aprendizaje por refuerzo (*Q-Learning*, por ejemplo) resultan poco efectivos.

En DQN, el agente es una red convolucional que recibe como entradas el estado del entorno en forma de imágenes en bruto. Entrenar el agente es entrenar la red convolucional para que, dado un estado representado con imágenes, la red determine la mejor acción.

El objetivo de *DeepMind* era entrenar con DQN agentes que resolvieran los juegos de *Atari 2600*, utilizando únicamente salidas de vídeo RGB, refuerzos y señales de terminación provistos por el *framework* ALE [15].

Aplicar técnicas de *Deep Learning* al aprendizaje por refuerzo tradicional plantea algunos retos. Hasta el momento, los métodos de DL han sido explotados para resolver tareas de aprendizaje supervisado y no supervisado, en las que los ejemplos de entrenamiento son independientes. Sin embargo, los datos en aprendizaje por refuerzo comúnmente están correlacionados.

Para solucionar ese problema, en el artículo proponen el concepto *Experience Replay*. Consiste en incluir en DQN una memoria, denominada *memory replay*, en la que almacenan experiencias del tipo *<estado, acción, refuerzo, nuevo estado>*. Estas experiencias se producen durante la ejecución de DQN y son utilizadas para el entrenamiento de la red convolucional.

4.3.1.1 DQN con *Experience Replay*

La meta de DQN es entrenar una red convolucional para que, dado un estado de la partida φ_t, genere una buena estimación del refuerzo futuro para toda acción a permitida, $Q(\varphi_t, a)$. Por tanto, la red actuaría como una función aproximadora de valores Q, y sustituiría a las tablas Q propias de algoritmos de aprendizaje por refuerzo tradicionales (*Q-Learning*), que son muy poco eficaces en entornos con un espacio de estados excesivamente grande.

El pseudocódigo de DQN presentado en el artículo ha sido adaptado y se presenta en la Tabla 4:

Inicializar aleatoriamente de los pesos de la CNN.
Inicializar la *memory replay R* de capacidad *N*.
FOR $episodio\ in\ [1 \dots M]$:
    Inicializar secuencia de imágenes $s_1 = \{x_1\}$
    Procesar secuencia $\varphi_1 = \varphi(s_1)$
    FOR $t\ in\ [1 \dots T]$:
        Con probabilidad ϵ se elige una acción a_t aleatoria
        En caso contrario se elige la acción $a_t = \arg\max_a Q^*(\varphi(s_t), a)$
        Ejecutar a_t y recibir el refuerzo r_t y la imagen x_{t+1}
        Inicializar secuencia $s_{t+1} = s_t, a_t, x_{t+1}$
        Procesar secuencia $\varphi_{t+1} = \varphi(s_{t+1})$
        Almacenar experiencia $(\varphi_t, a_t, r_t, \varphi_{t+1})$ en *R*
        Elegir un subconjunto aleatorio de experiencias $(\varphi_j, a_j, r_j, \varphi_{j+1})$ de *R*
        Por cada experiencia del conjunto, obtener y_j:
            $y_j = r_j$, si φ_{j+1} es terminal
            $y_j = r_j + \gamma \cdot max_a, Q(\varphi_{j+1}, a')$, si φ_{j+1} no es terminal
        Aplicar descenso del gradiente en CNN, dado y_j y $Q(\varphi_j, a_j)$
    END FOR
END FOR

Tabla 4. Pseudocódigo de DQN

De este procedimiento pueden extraerse los siguientes comentarios:

- $\boldsymbol{s_t}$ contiene la secuencia de imágenes observadas (x_j) y acciones elegidas (a_j) desde el inicio del episodio.

$$s_t = \{x_1, a_1, x_2, a_2, \dots, x_{t-1}, a_{t-1}, x_t\}$$

- $\boldsymbol{\varphi(s_t)}$ es una función que transforma la secuencia s_t en un estado φ_t. φ_t contiene una secuencia de las últimas k imágenes observadas, a las que se le aplican transformaciones (por ejemplo, conversión a escala de grises). φ_t es la entrada de la red convolucional.

$$\varphi_t = \varphi(s_t) = \varphi(\{\dots, x_{t-k+1}, a_{t-k+1} \dots, x_{t-1}, a_{t-1}, x_t\}) = \{x'_{t-k+1}, \dots, x'_{t-1}, x'_t\}$$

- Se sigue una estrategia ***ϵ-greedy*** para elegir la acción a ejecutar. La acción puede ser aleatoria, o aquella que tenga asociado el mayor valor Q de salida, al pasar el estado φ_t como entrada de la red (véase la Ilustración 10).

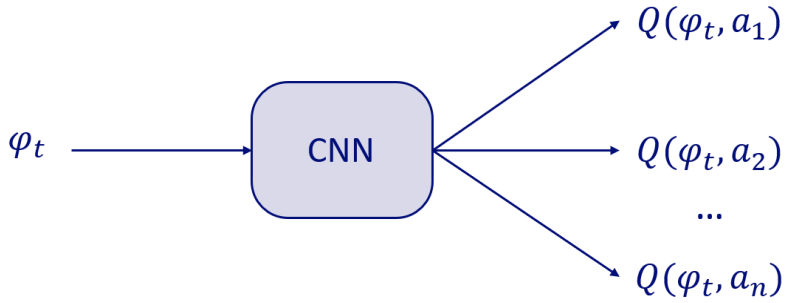


Ilustración 10. Entradas y salidas de la red convolucional

- El entrenamiento de la red se realiza con tuplas de experiencia $(\varphi_j, a_j, r_j, \varphi_{j+1})$ que se almacenaron en la *memory replay* desde el inicio del entrenamiento. Los elementos de la tupla son:

  - φ_j, estado del episodio en el instante j.
  - a_j, acción ejecutada en el emulador en el instante j, dado el estado φ_j.
  - r_j, refuerzo obtenido en el instante j al ejecutar la acción a_j sobre el estado φ_j.
  - φ_{j+1}, nuevo estado tras ejecutar la acción a_j.

  El ajuste de los pesos de la red se realiza con un algoritmo basado en el descenso del gradiente del error. Dada una tupla de experiencia $(\varphi_j, a_j, r_j, \varphi_{j+1})$, el error viene determinado por la diferencia al cuadrado entre la salida obtenida $Q(\varphi_j, a_j)$, y la salida esperada y_j, que es:

  $$y_j = r_j, \text{ si } \varphi_{j+1} \text{ es terminal, o}$$

  $$y_j = r_j + \gamma \cdot max_a, Q(\varphi_{j+1}, a'), \text{ si } \varphi_{j+1} \text{ no es terminal}$$

4.3.1.2 Experimentación con DQN.

La experimentación de DQN se hizo sobre algunos videojuegos de *Atari 2600* porque son entornos con un espacio de estados enorme para los que las técnicas tradicionales de aprendizaje por refuerzo han demostrado ser insuficientes.

Los juegos sobre los que se experimentó con DQN fueron *Beam Rider*, *Breakout*, *Enduro*, *Pong*, *Q*Bert*, *Seaquest* y *Space Invaders*. Para todos ellos, la arquitectura de DQN y sus parámetros no fueron modificados. Los autores también utilizaron otros algoritmos de aprendizaje para comparar los resultados.

La arquitectura de la red convolucional de DQN presenta las siguientes características:

- **Capa de entrada.**

  La entrada es una imagen de dimensión 84x84x4, producida por la función φ. Son 4 imágenes de dimensión 84x84 píxeles, que son el resultado de una serie

de transformaciones: (1) conversión de RGB a escala de grises de las imágenes originales de 160x210 píxeles, (2) reducción a 84x110 píxeles y (3) recorte a 84x84 píxeles.

En la Ilustración 11 se muestra un ejemplo de procesamiento de la entrada:

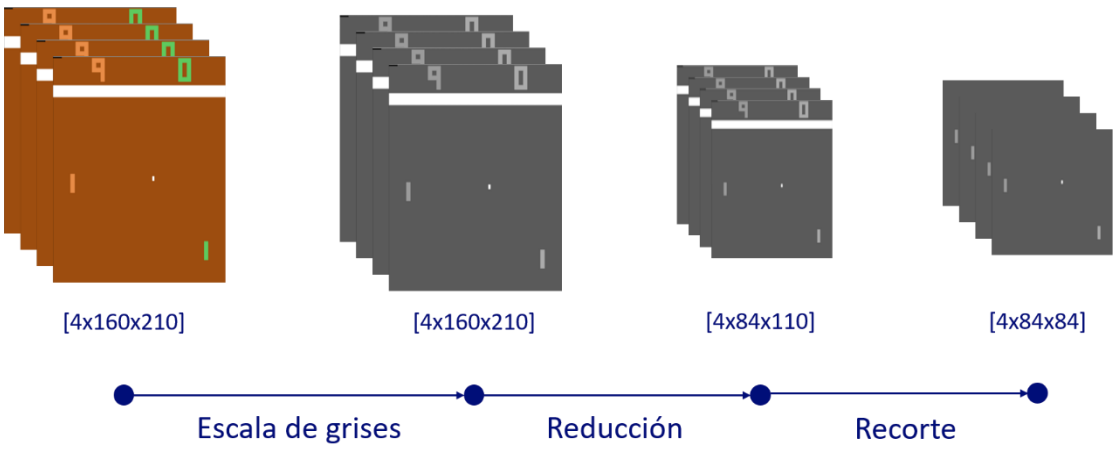


Ilustración 11. Transformación de las imágenes de un estado del juego 'Pong'

Las 4 imágenes originales se corresponden con los 4 *frames* más recientes de la partida, utilizando la técnica *frame-skipping*, que se describirá más adelante.

- **Capa de salida.**

  El número de salidas va de 4 a 18, según el número de acciones que ofrezca el emulador para un determinado juego. Las salidas son los valores Q asociados a cada acción, dada la entrada.

- **Capas internas.**

  Contiene 2 capas de convolución: la primera de 16 filtros de dimensión 8x8 y la segunda de 32 filtros de dimensión 4x4; ambas con función de activación no lineal. La segunda capa de convolución se conecta con una red estándar (*fully-conected*) de 256 neuronas, que produce la salida de la red.

Las consideraciones de los autores para la experimentación son:

1. ϵ-greedy con $\epsilon_0 = 1$, que desciende linealmente a 0.1 hasta el millón de *frames*.
2. *Memory replay* con capacidad para 1 millón de experiencias recientes.
3. Los refuerzos negativos se transforman a -1 y los positivos, a 1, para igualar los refuerzos entre distintos juegos.
4. Utilización de una técnica denominada *frame-skipping*, que consiste en seleccionar una acción cada k *frames* y repetir esa misma acción los siguientes $k - 1$ *frames*. El *frame-skipping* permite reducir el tiempo de ejecución sin

perder efectividad del aprendizaje. Para algunos juegos se utilizo $k = 3$ y para otros $k = 4$.

Los experimentos se ejecutaron sobre máquinas que disponían de potentes tarjetas gráficas, capaces de soportar la carga de trabajo que conllevar entrenar una red de neuronas.

4.3.1.3 Conclusiones

Los resultados determinaron que DQN había conseguido la mejor puntuación para 6 de los 7 juegos probados. Esto demostró que la combinación de redes de neuronas profundas con *reinforcement learning* es factible para entrenar agentes a partir de grandes volúmenes de datos sensoriales de alta dimensionalidad.

4.3.2 *Human-level control through deep reinforcement learning*

Los autores redactaron otro artículo de tipo experimental titulado *Human-level control through deep reinforcement learning* [16] en el que probaron DQN con un conjunto mucho mayor de videojuegos de Atari. Tuvieron las mismas consideraciones que en el otro artículo, pero cambiaron ligeramente la arquitectura de la red convolucional. Contiene 3 capas de convolución: la primera de 32 filtros de dimensión 8x8, la segunda de 64 filtros de dimensión 4x4 y la tercera de 64 filtros de dimensión 3x3; todas con función de activación no lineal. La segunda capa de convolución se conecta con una red estándar (*fully-conected*) de 512 neuronas, que produce la salida de la red.

Aplicaron la misma arquitectura y los mismos parámetros de DQN para todos los videojuegos de la experimentación y obtuvieron resultados muy positivos en una gran parte de ellos.

4.3.3 *Classical Planning with Simulators: Results on the Atari Video Games*

En este artículo [17] se evalúa un algoritmo de planificación para resolver juegos de Atari utilizando el *framework* ALE propuesto por Bellemare y su equipo en su artículo de 2013 [15].

ALE permite abordar la tarea de dos formas: con planificación (*planning*) y con aprendizaje por refuerzo (*reinforcement learning*). Los autores del artículo se centraron en el estudio de las técnicas de *planning*, y dejaron a un lado las de *reinforcement learning*.

El problema de utilizar planificadores clásicos en ALE es que no existe una codificación del dominio al estilo de PDDL y tampoco se conoce la meta a alcanzar. Por ello, los autores se centraron en algoritmos de planificación novedosos, relacionados con los métodos de búsqueda no informados. Estos algoritmos son capaces de trabajar eficientemente en grandes espacios de estados sin necesidad de conocer metas, costes o transiciones entre estados. IW es el algoritmo de planificación evaluado en el artículo.

4.3.3.1 IW

Iterated Width (IW) es un algoritmo que, dado un problema de planificación, encuentra un plan de acciones que soluciona el problema. Para que el problema pueda ser resuelto con IW, es necesario que pueda definirse como un espacio de estados, en el cual se cumplan dos condiciones:

1. Cada **estado** s (véase la Ilustración 12) se representa como un vector de n variables discretas X_i.
2. Cada **variable** X_i presenta un dominio de posibles valores x_i.

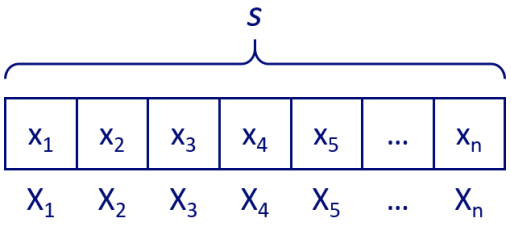


Ilustración 12. Representación de un estado del problema en IW

La asignación de un valor x_i a una variable X_i, $X_i = x_i$, se conoce como **átomo**.

IW es una secuencia de llamadas a $IW(i)$, donde $i = 1, 2, 3$ ...., que finaliza cuando se alcanza una meta. $IW(i)$ es un algoritmo de búsqueda ciega en amplitud, con la diferencia de que, al expandir un nodo, a los nodos hijos generados se les asigna un valor de *novedad*, y son podados si no aprueban un *test de novedad*.

La *novedad* determina cómo de "nuevo" es un nodo (estado). La novedad de un estado es j si es el primer estado generado en la búsqueda que hace cierta una tupla mínima de j átomos. Novedades bajas pertenecen a estados "nuevos"; novedades altas pertenecen a estados "antiguos". Un estado repetido tendrá la máxima novedad, $n + 1$, siendo n el número de variables del estado.

Para $IW(i)$, los nodos que tengan una novedad mayor que i se consideran nodos repetidos que deben ser podados. Por ello, se dice que suspenden el test de novedad.

En el anexo, se incluye un ejemplo sencillo sobre el funcionamiento de IW.

4.3.3.2 IW para ALE

Los autores implementaron IW dentro del *framework* ALE para poder experimentar con él en el ámbito de los videojuegos de *Atari 2600*. Sin embargo, realizaron algunas modificaciones de IW original:

1. Al expandir un nodo, los hijos se generan en orden aleatorio.

2. Limitación a $IW(1)$ para evitar el crecimiento excesivo del tamaño del árbol de búsqueda para $i > 1$.
3. Al no existir una meta definida, se establece un límite de nodos generados como criterio de parada.

Los estados del problema cuentan con 128 variables, que se corresponden con los 128 Bytes de la memoria RAM aportados por el emulador de ALE. Cada variable admite valores comprendidos entre 0 y 7.

Durante la partida de un videojuego, el agente realiza iterativamente un procedimiento similar al de la Ilustración 9:

1. Realiza una copia del estado actual (memoria RAM) de la partida localizado en el emulador, para no perderlo.

2. Ejecuta $IW(1)$, siendo el nodo raíz el estado actual de la partida. $IW(1)$ genera un árbol de estados que representa la simulación de un abanico de secuencias de futuras acciones desde el estado actual. Cada nodo s tiene asociado un refuerzo acumulado $R(s)$ que es la suma del refuerzo del nodo padre s' y del refuerzo actual obtenido al ejecutar una acción a.

$$R(s) = R(s') + r(a, s')$$

3. Cuando el árbol ha alcanzado el límite de nodos permitidos, la simulación termina. La mejor secuencia de acciones futuras se corresponde con la rama del árbol que ha acumulado mayor refuerzo.

4. Restaura en el emulador el estado actual de la partida, guardado en el paso 1.

5. Escoge la mejor acción para ejecutar, que se corresponde con la primera acción de la rama con mayor refuerzo acumulado. Envía la acción elegida al emulador para que la ejecute y este le proporciona el nuevo estado y el nuevo refuerzo obtenido. Si la partida no ha finalizado, vuelta al paso 1.

El código de esta versión de IW se encuentra disponible en un repositorio *GitHub* [20], implementado dentro del paquete de ALE.

4.3.3.3 Experimentación

Para la experimentación, los autores definieron los siguientes parámetros:

- Factor de descuento $\gamma = 0.995$.
- Límite de 150.000 *frames* simulados con $IW(1)$.
- 10 episodios por cada uno de los juegos probados.
- Ejecución de $IW(1)$ cada 5 *frames*. La acción elegida se repite los 4 *frames* siguientes, para aligerar el procesamiento.

Los experimentos fueron ejecutados en una máquina con procesador Intel Xeon [45] y 64 GB de memoria RAM.

IW fue comparado en tiempo y puntuación media con otros algoritmos de planificación. Se utilizaron 54 juegos de *Atari 2600* para la experimentación y se determinó que IW obtenía la mejor puntuación media en 27 de los 54 juegos, y que conseguía el menor tiempo de ejecución en 18 de los 54 juegos.

4.3.4 *Evolving simple programs for playing Atari games*

Se trata de un artículo muy reciente [18] en el que se propone un método denominado *Cartesian Genetic Programming* (CGP) para abordar la tarea de resolver los juegos de Atari.

CGP es una técnica de computación evolutiva que ha demostrado ser muy útil para resolver tareas de procesamiento o filtrado de imágenes en diversos ámbitos. Dado que el dominio de los juegos de Atari cuenta con ALE [15], un *framework* que facilita el desarrollo de agentes basados en técnicas de IA y que proporciona el estado de las partidas en forma de matrices de píxeles, los investigadores consideraron que CGP podría encajar en este dominio.

El objetivo de los investigadores era generar programas de computador que resolviesen partidas de los videojuegos de Atari tomando decisiones basándose en el procesamiento de matrices de píxeles de entrada.

4.3.4.1 CGP

CGP es una forma de programación genética en la que los individuos de la población representan programas de computador a través de grafos dirigidos y, normalmente, acíclicos. Los nodos del grafo se conectan por medio de coordenadas cartesianas. Además, cada nodo tiene asociada una función, comúnmente matemática o estadística (ADD, STDDEV, FLOOR, etc.), que recibe como entrada los datos de salida de los nodos que se conectan a él.

Las conexiones entre los nodos crean canales por los cuáles las entradas del programa son procesadas hasta obtener una salida. Los nodos que quedan desconectados del grafo no participan en el procesamiento de las entradas.

La evolución de un individuo conlleva un cambio en las funciones asociadas a los nodos y en las conexiones entre ellos, por lo que el programa codificado en el individuo cambia la forma de procesar las entradas.

4.3.4.2 Especificación del genoma para el dominio de Atari

Para este trabajo, los investigadores establecieron la codificación de los individuos de la siguiente manera:

- Cada gen es un valor numérico real, y su valor está restringido al intervalo $[0, 1]$.

- El grafo del individuo está constituido por $N = C + n_{input}$ nodos, donde C es el número de nodos intermedios y n_{input} es el número de nodos de entrada.
- Cada nodo n intermedio está formado por 4 genes:
  - Los genes x e y se utilizan para determinar qué nodos se conectan al nodo n.
  - El gen de función f representa el índice a una función de la lista de funciones establecida.
  - El gen p representa un parámetro de entrada de la función asociada al nodo n.
- El genoma (véase la Ilustración 13) tiene un tamaño fijo y consta de $G = n_{output} + 4 \cdot C$ genes, donde n_{output} es el número de genes de salida.
- El valor de un gen de salida se utiliza para determinar el nodo (de entrada o intermedio) conectado a la salida del programa.

Los investigadores establecieron que el número de nodos intermedios sería $C = 60$. También determinaron que el número de nodos de entrada sería $n_{input} = 3$, correspondientes a las componentes Rojo, Azul y Verde de la matriz de píxeles RGB de un instante de la partida. En cuanto al número de genes de salida n_{output}, coincide con la cardinalidad del conjunto mínimo de acciones del videojuego, que varía entre 4 y 18.

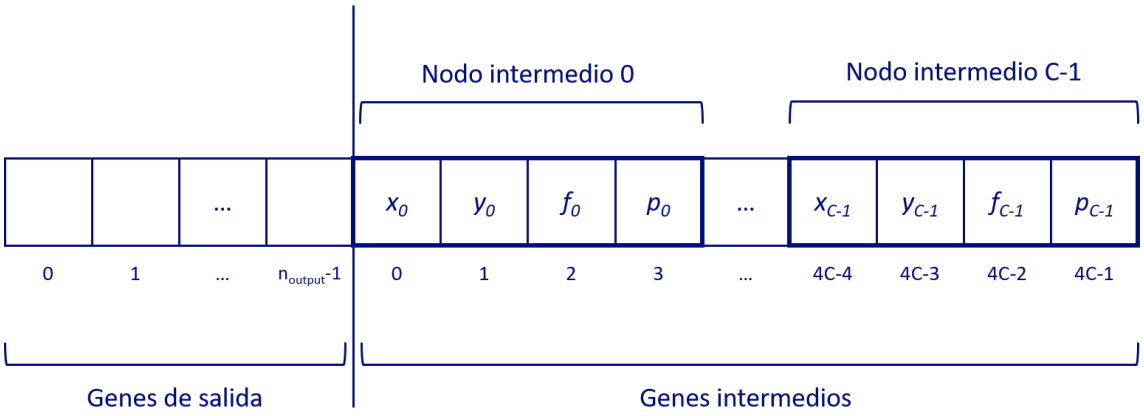


Ilustración 13. Representación del genoma del individuo

La población consta de un único individuo, y evoluciona según el siguiente procedimiento:

1. Inicialización del individuo de la población inicial. Cada gen del genoma adopta un valor real comprendido entre 0 y 1.

2. Generación de un número λ de descendientes aplicando mutaciones sobre el individuo de la población. Los investigadores escogieron un valor $\lambda = 9$.
3. Evaluación de todos los individuos. Si alguno de los descendientes supera el valor de *fitness* del individuo de la población, el descendiente lo sustituye.
4. Si se ha superado un número de evaluaciones preestablecido, el procedimiento finaliza y el individuo de la población es el programa evolucionado. En caso contrario, se vuelve al paso 2.

La función de *fitness* con la que se evalúa cada individuo consiste en ejecutar una partida de un juego de Atari, utilizando en cada instante de la partida el programa codificado en el genoma. El programa actuaría como un agente que, dada la imagen del estado de la partida, determina la acción a ejecutar. El valor de *fitness* es la puntuación obtenida por el programa en la partida.

4.3.4.3 Discusión

Se experimentó con CGP en 61 videojuegos, y los resultados fueron comparados con otras técnicas aplicadas hasta el momento en el dominio de Atari. En 9 de los juegos CGP obtuvo la puntuación más elevada, lo cual fue todo un logro teniendo en cuenta que este trabajo representaba una primera aproximación y que los programas evolucionados eran relativamente sencillos. En los demás videojuegos CGP proporcionó resultados mejorables, aunque prometedores.

CGP se consolida como una técnica competente en el dominio de Atari. La principal ventaja de CGP es la sencillez de los programas y la rapidez con que evolucionan. Sin embargo, todavía necesite mejorar en algunos aspectos. Por ejemplo, se observó que algunos programas evolucionados seguían estrategias muy básicas basados en esconderse o golpear constantemente. El proceso de evolución no conseguía refinar estas estrategias, por lo que los investigadores tratarán de mejorarlo en futuros trabajos.

5 Planteamiento de la cuestión

Como se ha podido comprobar en los artículos analizados en los casos de estudio sobre DQN [14] e IW [17], existen dos líneas de investigación consolidadas en el dominio de los juegos de Atari: *planning* y *reinforcement learning*.

Ambos enfoques son completamente diferentes:

- Los algoritmos de *planning* como IW deciden en cada momento la siguiente acción a ejecutar realizando una simulación de la partida desde el estado actual. Para ello, utilizan el espacio de acciones permitidas y despliegan un árbol donde cada rama representa una posible línea temporal futura. Cuando termina la simulación, se conoce la secuencia de acciones futuras que proporciona el mayor refuerzo estimado. La primera acción de esa secuencia será la que se ejecute.
- Los algoritmos de *reinforcement learning* como DQN deciden qué acción ejecutar en cada momento por medio de un controlador (en caso de DQN, sería una red convolucional), que necesita ser entrenado previamente. El entrenamiento requiere una gran cantidad de episodios (partidas) para que el controlador aprenda a sortear cualquier situación que pueda darse durante la partida.

Debido a estas diferencias, cada enfoque tiene sus propias ventajas y desventajas. Un punto a favor de las técnicas de *planning* es que proporcionan, por lo general, mejores resultados que las técnicas de *reinforcement learning*, como puede observarse en la Tabla 5. Como punto negativo, los resultados dependen en gran medida de la exhaustividad de la simulación. Esto implica que el *planning* no es una buena aproximación para dotar de inteligencia a un agente que debe comportarse en tiempo real, porque averiguar la siguiente acción podría necesitar mucho tiempo de simulación y eso afectaría al desarrollo de la partida.

Juego	Puntuacion - Técnicas	
	IW	Redes de DQN
Breakout	384	401.2 (±26.9)
Pong	21	18.9 (±1.3)
Frostbite	902	328.3 (±250.5)
Asteroids	153400	1629 (±542)
Beam Rider	9108	6846 (±1619)
Space Invaders	2877	1976 (±893)
*Q*Bert*	3705	10596 (±3294)
Ice Hockey	55	-1.6 (±2.5)

Tabla 5. Comparativa de puntuaciones IW – DQN para algunos juegos de Atari [17] [16]

Por el contrario, los métodos de *reinforcement learning* generan controladores capaces de determinar la mejor acción en cada situación de forma inmediata, por lo que serían útiles para agentes inteligentes que no comprometan la fluidez de la partida. Como punto negativo, el entrenamiento del controlador necesita una buena configuración de parámetros y mucho tiempo de aprendizaje, y aun así es posible que para determinados juegos el entrenamiento no sea lo suficientemente eficaz.

La cuestión que se plantea en este proyecto es si es posible combinar estos dos enfoques para mejorar la forma de resolver los juegos de la consola *Atari 2600*. En concreto, se propone utilizar una red convolucional entrenada con el algoritmo DQN (del caso de estudio [14]) para ayudar al algoritmo IW (del caso de estudio [17]) a decidir qué acciones son más apropiadas a lo largo de la partida. De ahora en adelante, a este planteamiento se le denominará ***combinación IW+DQN***.

La idea guarda cierto parecido con los *ensembles* de clasificadores de tipo *stacking* [46], que abordan tareas de clasificación utilizando de forma conjunta distintos algoritmos para conseguir predicciones de mayor confianza que utilizando dichos algoritmos de forma individual.

Las posibilidades que se presentan con este planteamiento son varias. Con la ayuda de una red entrenada con DQN, las simulaciones de IW tal vez podrían ser menos exhaustivas, y con ello, podría reducirse considerablemente el tiempo de ejecución sin perder efectividad. Además, con la colaboración de IW, tal vez podría ser suficiente con una red menos entrenada, lo que supondrían un ahorro de tiempo de entrenamiento.

En resumen, la cuestión que se plantea es si la combinación IW+DQN es más efectiva y eficiente a la hora de resolver los juegos de *Atari 2600* que las dos técnicas por separado.

6 Desarrollo de la cuestión

En este apartado se desarrollará la combinación IW+DQN planteada en el apartado anterior. Para ello, será necesaria una metodología de desarrollo *software*. Se basará en el *modelo en cascada* [19] (véase la Ilustración 14), que divide el proyecto *software* en etapas que se ejecutan secuencialmente siguiendo un orden lógico. Las etapas comunes son:

1. **Análisis.** Se establecen los objetivos y se analizarán las necesidades que deberá cubrir el *software*, formalizándolas como requisitos.
2. **Diseño.** Teniendo en cuenta el análisis previo, se definen la arquitectura y comportamiento teóricos del *software*, por medio de diversos diagramas.
3. **Implementación.** Se implementa el *software* siguiendo el diseño especificado en la etapa anterior.
4. **Pruebas.** Con el *software* ya implementado, se comprueba que funciona correctamente y que cumple los requisitos.
5. **Verificación.** El producto *software* ya probado se entrega al usuario final, que lo ejecuta y da su visto bueno.
6. **Mantenimiento.** Se corrigen errores que puedan surgir después de la entrega del producto al usuario final.

Si se encontrase algún error o imprevisto en algún punto del desarrollo *software*, la metodología permite volver a etapas anteriores para solventarlo.

Se ha escogido este método por varias razones:

- Es conocido y fácil de entender y utilizar con el paradigma de Programación Orientada a Objetos (POO).
- Es muy adecuado para proyectos sencillos y en los que no hay riesgo de que se modifiquen los objetivos planteados inicialmente.
- La división por etapas secuenciales facilita la planificación del proyecto.

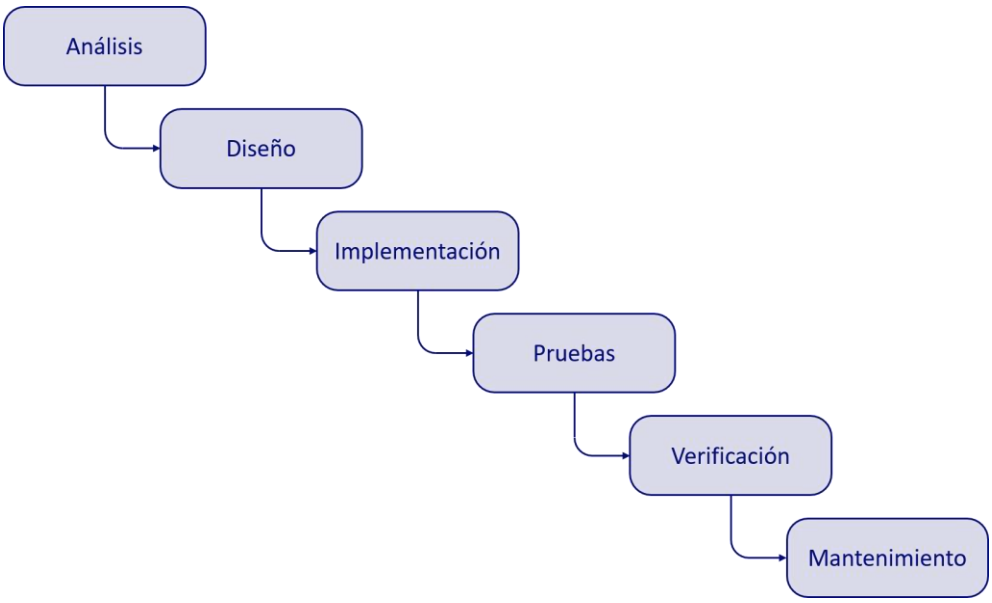


Ilustración 14. Metodología en cascada

6.1 Análisis

Para desarrollar la combinación IW+DQN se tomarán como base dos códigos ya existentes:

- El código *C++* extraído del repositorio ***ALE-Atari-Width*** de *Github* [20]. Implementa el algoritmo de planificación IW dentro del *framework* ALE [10]. Fue construido y utilizado por los autores del caso de estudio sobre IW [17] para experimentar con el algoritmo.
- El código *Python* extraído del repositorio ***DQN-tensorflow*** de *Github* [21]. Implementa el algoritmo DQN sobre las librerías *Gym* [11] y *Tensorflow* [12]. Permite guardar el modelo de red entrenado en un fichero para que pueda ser utilizado en cualquier momento. Se ha escogido este código entre muchas otras implementaciones de DQN porque es el más fiel a las especificaciones del equipo de *DeepMind* en su artículo sobre DQN [16].

Antes de definir las funcionalidades del futuro *software* por medio de requisitos de usuario y *software*, es necesario analizar con mayor detenimiento los códigos base mencionados, sobre todo *ALE-Atari-Width*.

6.1.1 Análisis de los códigos

ALE-Atari-Width dispone de un conjunto de parámetros que se pasan como entrada por la línea de comandos. De todos ellos, los más relevantes son:

- ***game*.** Permite al usuario elegir el juego de *Atari 2600* que se emulará.
- ***game_controller.*** Permite al usuario elegir el controlador de la partida.

- ***player_agent.*** Permite al usuario escoger el tipo de agente que resolverá una partida del videojuego elegido. En el código hay diversos agentes: *SearchAgent*, *RandomAgent*, etc. *SearchAgent* es el agente que utiliza un algoritmo de planificación (búsqueda) para abordar la partida. *RandomAgent* sería el agente que ejecuta acciones aleatorias en cada *frame* de la partida.
- ***search_method.*** En caso de escoger el agente *SearchAgent*, este parámetro permite elegir el algoritmo de planificación que utilizará el agente durante la partida. Hay varios, pero para este proyecto solo interesa IW.

Cuando se ejecuta el código, se inicializa el controlador. Este se encarga, entre otras cosas, de cargar en el emulador el videojuego especificado con el parámetro *game*. También se encarga de inicializar el tipo de agente definido con el parámetro *player_agent*. A partir de ahí, el controlador inicia un proceso iterativo en el que recibe observaciones del estado de la partida y llama al agente para que decida la acción a ejecutar, dado ese estado.

En la Ilustración 15 se muestra un esquema del funcionamiento de *ALE-Atari-Width* cuando el agente (*player_agent*) es *SearchAgent* y el algoritmo de planificación (*search_method*) es IW:

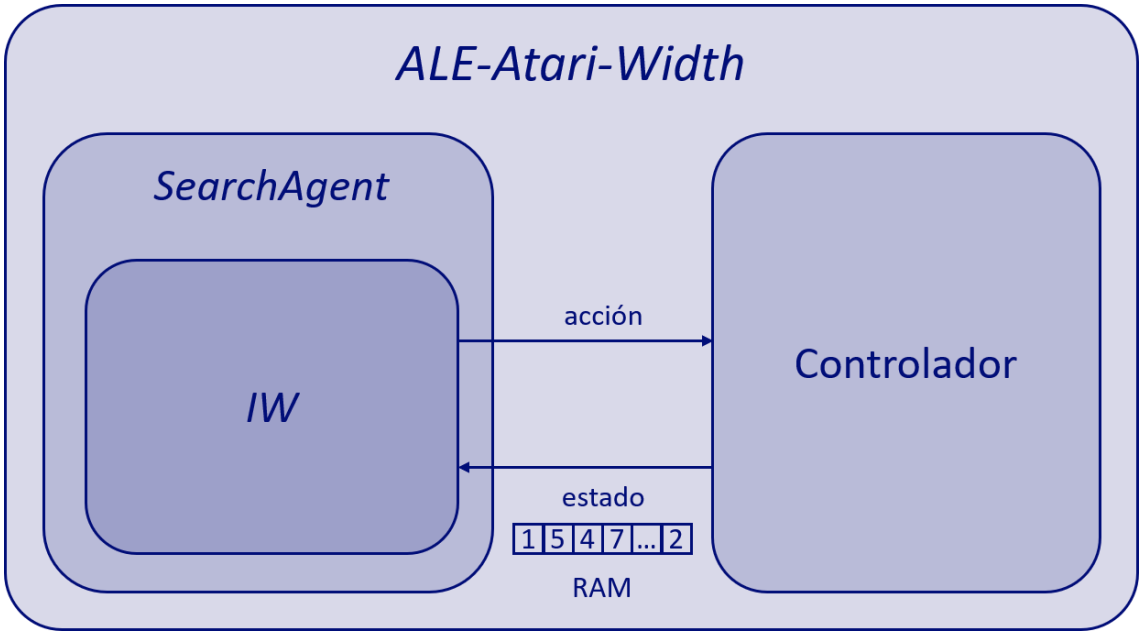


Ilustración 15. Esquema del funcionamiento ALE-Atari-Width

Entendiendo todo esto, desarrollar la combinación IW+DQN planteada es sinónimo de desarrollar un nuevo agente, que podría implementarse dentro del código *ALE-Atari-Width* junto con los ya existentes. A este nuevo agente se le denominará ***CombinationAgent***.

En la Ilustración 16 puede verse un esquema a alto nivel de la idea. El sistema planteado combinaría los subsistemas *DQN-tensorflow* y *ALE-Atari-Width*. Durante la ejecución, el controlador proporcionaría al agente *CombinationAgent* el estado de la partida. El agente procesaría el estado y se lo pasaría a IW y a la red de DQN, para que tomen una decisión por separado. Por último, el agente escogería la acción definitiva basándose en las decisiones de ambas técnicas y la enviaría al controlador para que la ejecute.

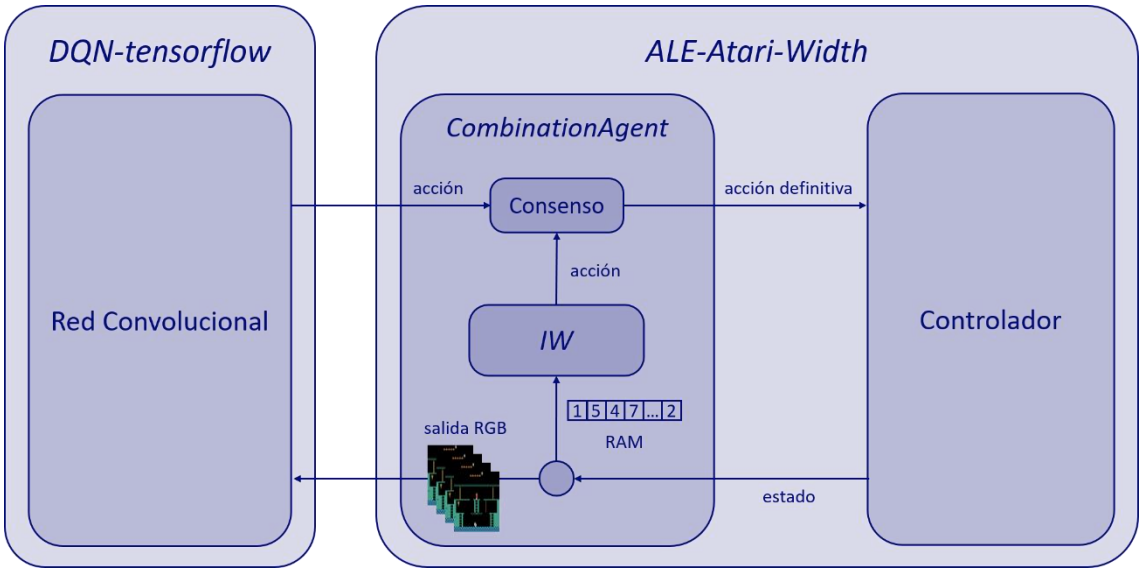


Ilustración 16. Esquema de CombinationAgent en relación con el resto del sistema software

6.1.2 Requisitos de usuario

A continuación, se adjuntan las tablas de requisitos de usuario, que cuentan con los siguientes campos:

- **Identificador "RU-XXX".** De las siglas *Requisito de Usuario*.
- **Descripción.** Definición del requisito.
- **Prioridad.** Determina el grado preferencia del requisito a ser aplicado.
- **Necesidad.** Determina el grado de importancia del requisito dentro del sistema.

RU-001			
Nombre	Integración del *CombinationAgent*.		
Descripción	El sistema deberá integrar un nuevo agente que permita que un algoritmo de planificación como IW y una red convolucional de DQN trabajen conjuntamente para resolver una partida de cualquier videojuego de *Atari 2600*.		
Prioridad	☒ Alta	Necesidad	☒ Alta

RU-001			
	☐ Media ☐ Baja		☐ Media ☐ Baja

Tabla 6. RU-001

RU-002			
Nombre	Elección de *CombinationAgent*.		
Descripción	El sistema deberá permitir al usuario elegir el agente *CombinationAgent*, entre todos los posibles agentes ya existentes.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 7. RU-002

RU-003			
Nombre	Influencia de la red de DQN en *CombinationAgent*.		
Descripción	Si el usuario escoge el agente *CombinationAgent*, el sistema deberá permitir al usuario elegir el grado de influencia de la red de DQN en las decisiones durante la partida.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 8. RU-003

RU-004			
Nombre	Elección del algoritmo de planificación.		
Descripción	Si el usuario escoge el agente *CombinationAgent*, el sistema deberá permitir al usuario elegir el algoritmo de planificación.		
Prioridad	☐ Alta ☒ Media ☐ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 9. RU-004

RU-005			
Nombre	Elección del modo de combinación.		
Descripción	Si el usuario escoge el agente *CombinationAgent*, el sistema deberá permitir al usuario elegir entre distintos modos de combinación.		
Prioridad	☐ Alta ☒ Media ☐ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 10. RU-005

RU-006			
Nombre	Modo de consenso.		
Descripción	El agente *CombinationAgent* deberá contar con un modo en el que, durante la partida, las decisiones sean consensuadas entre la red de DQN y el algoritmo de planificación.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 11. RU-006

RU-007			
Nombre	Modo de probabilidad.		
Descripción	El agente *CombinationAgent* deberá contar con un modo en el que, en cada *frame* de la partida, las decisiones correspondan únicamente a la red de DQN o al algoritmo de planificación, según un factor aleatorio.		
Prioridad	☐ Alta ☒ Media ☐ Baja	Necesidad	☐ Alta ☐ Media ☒ Baja

Tabla 12. RU-007

RU-008			
Nombre	Interacción con la red de DQN.		
Descripción	El agente *CombinationAgent* deberá ponerse en contacto con la red de DQN para la toma de decisiones, siempre que se requiera.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 13. RU-008

6.1.3 Requisitos *software*

6.1.3.1 Requisitos funcionales

A continuación, se adjuntan las tablas de requisitos *software* funcionales, que cuentan con los siguientes campos:

- **Identificador "RS-FXXX".** De las siglas *Requisito Software Funcional*.
- **Descripción.** Definición del requisito.
- **Prioridad.** Determina el grado preferencia del requisito a ser aplicado.
- **Necesidad.** Determina el grado de importancia del requisito dentro del sistema.

RS-F001			
Nombre	Reutilización del parámetro *player_agent*.		
Descripción	El sistema permitirá que el parámetro de entrada *player_agent* pueda adoptar el valor "combination_agent", que permitirá inicializar el agente *CombinationAgent*.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 14. RS-F001

RS-F002			
Nombre	Creación del parámetro *ratio*.		
Descripción	El sistema incorporará un nuevo parámetro de entrada denominado *ratio*, que se utilizará únicamente cuando el usuario haya escogido el agente *CombinationAgent*. *ratio* podrá adoptar los valores reales comprendidos en el intervalo [0,1]. El valor por defecto de *ratio* será 0.5.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 15. RS-F002

RS-F003			
Nombre	Reutilización del parámetro *search_method*.		
Descripción	El sistema aprovechará el parámetro de entrada *search_method* para que el usuario pueda elegir el algoritmo de planificación si ha escogido el agente *CombinationAgent*.		
Prioridad	☐ Alta ☒ Media ☐ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 16. RS-F003

RS-F004			
Nombre	Ejecución del agente *CombinationAgent*.		
Descripción	Si el usuario elige el agente *CombinationAgent*, el controlador del sistema deberá llamar a dicho agente en cada *frame* de la partida para que decida las acciones y que el controlador las ejecute.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 17. RS-F004

RS-F005			
Nombre	Reutilización del parámetro *sim_steps_per_node*.		
Descripción	El sistema aprovechará el parámetro de entrada *sim_steps_per_node* que define cada cuantos *frames* el agente decidirá una nueva acción. En los *frames* intermedios, el agente se limitará a repetir la última acción decidida.		
Prioridad	☐ Alta ☐ Media ☒ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 18. RS-F005

RS-F006			
Nombre	Creación del parámetro *modo*.		
Descripción	El sistema incorporará un nuevo parámetro de entrada denominado *modo*, que se utilizará únicamente cuando el usuario haya escogido el agente *CombinationAgent*. *modo* definirá la forma en que cooperarán la red de DQN y el algoritmo de planificación. El parámetro adoptará 2 valores: “consensus” y “probability”. El valor por defecto será “consensus”.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 19. RS-F006

RS-F007	
Nombre	Modo “consensus”.
Descripción	El agente *CombinationAgent* dispondrá de un modo “consensus”. Con este modo, el agente hará lo siguiente en cada *frame* que le toque decidir acción: **1.** Se llamará al algoritmo de búsqueda para que realice la simulación y obtenga los refuerzos futuros asociados a cada posible acción. **2.** Se llamará a la red de DQN para que devuelva los valores Q asociados a cada posible acción. **3.** Se normalizarán tanto los refuerzos futuros como los valores Q, en un intervalo [0, 1].

RS-F007			
	4. Por cada acción, se realizará una suma ponderada del refuerzo futuro normalizado y del valor Q normalizado. El factor de ponderación será el valor del parámetro *ratio*. **5.** La acción consensuada será la que tenga mayor valor de la suma ponderada.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 20. RS-F007

RS-F008			
Nombre	Modo “probability”.		
Descripción	El agente *CombinationAgent* dispondrá de un modo “probability”. Con este modo, el agente dejará que elija acción o bien la red de DQN, o bien el algoritmo de planificación. La técnica escogida en cada *frame* dependerá de un factor aleatorio sesgado por el valor del parámetro *ratio*. **a)** Si le toca decidir a la red de DQN, la acción elegida será aquella que tenga asociado el máximo valor Q, después de haber ejecutado la red. **b)** Si le toca decidir a la técnica de búsqueda, la acción elegida será aquella que tenga asociado el máximo refuerzo futuro después de realizar la simulación.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 21. RS-F008

RS-F009			
Nombre	Envío de la entrada a la red de DQN.		
Descripción	Para ejecutar la red de DQN, será necesario escribir previamente la entrada de la red en un fichero. La entrada se corresponderá con 4 de las imágenes más recientes de la partida, en formato RGB. Las imágenes se escribirán en el fichero de más antiguas a más recientes.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 22. RS-F009

RS-F010			
Nombre	Recepción de las salidas de la red de DQN.		
Descripción	Una vez que ejecutada la red de DQN, las salidas generadas deberán ser escritas en un fichero. Las salidas serán una lista de valores numéricos reales.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 23. RS-F010

RS-F011			
Nombre	Reutilización del parámetro *game*.		
Descripción	El sistema aprovechará el parámetro de entrada *game* para que el agente *CombinationAgent*. De esta manera, el agente podrá llamar a la red de DQN entrenada para el videojuego definido con el parámetro *game*.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 24. RS-F011

RS-F012			
Nombre	Creación del parámetro *cnn_num_input*.		
Descripción	El sistema incorporará un nuevo parámetro de entrada denominado *cnn_num_input*, que se utilizará únicamente cuando el usuario haya escogido el agente *CombinationAgent*. *cnn_num_input* definirá el número de imágenes que se pueden pasar como entrada de la red de DQN. *cnn_num_input* podrá adoptar valores enteros positivos. El valor por defecto será 4.		
Prioridad	☐ Alta ☐ Media ☒ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 25. RS-F012

RS-F013			
Nombre	Creación del parámetro *cnn_action_repeat*.		
Descripción	El sistema incorporará un nuevo parámetro de entrada denominado *cnn_action_repeat*, que se utilizará únicamente cuando el usuario haya escogido el agente *CombinationAgent*. *cnn_action_repeat* definirá cada cuantos *frames* una imagen se pasará como entrada de la red. *cnn_action_repeat* podrá adoptar valores enteros positivos. El valor por defecto será 4.		
Prioridad	☐ Alta ☐ Media ☒ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 26. RS-F013

RS-F014	
Nombre	*Buffer* de imágenes.
Descripción	El agente *CombinationAgent* deberá encargarse de almacenar las imágenes (160x210 píxeles) asociadas a cada *frame* de la partida, para que sean utilizadas por la red. El tamaño del *buffer* estará limitado a *cnn_num_input* * *cnn_action_repeat*, para que solo se almacenen las imágenes más recientes y estrictamente necesarias.

RS-F014			
Prioridad	☐ Alta ☒ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 27. RS-F014

RS-F015			
Nombre	Creación del parámetro *search_frequency*.		
Descripción	El sistema incorporará un nuevo parámetro de entrada denominado *search_frequency*, que se utilizará únicamente cuando el usuario haya escogido el agente *CombinationAgent*. *search_frequency* definirá cada cuántos *frames* el agente utilizará únicamente el algoritmo de planificación definido con el parámetro *search_method*. *search_frequency* podrá adoptar los valores enteros positivos. El valor por defecto será 5.		
Prioridad	☐ Alta ☐ Media ☒ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 28. RS-F015

RS-F016			
Nombre	Procesamiento de imágenes.		
Descripción	Antes de ejecutar la red, las entradas pasadas por fichero deberán ser procesadas según lo establecido por los autores del caso de estudio sobre DQN [14]: **1.** Transformación de RGB a escala de grises. **2.** Reducción de tamaño de 160x210 a 110x84 píxeles. **3.** Recorte del área de interés (110x84 a 84x84 píxeles).		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☐ Alta ☒ Media ☐ Baja

Tabla 29. RS-F016

6.1.3.2 Requisitos no funcionales

A continuación, se adjuntan las tablas de requisitos *software* no funcionales, que cuentan con los siguientes campos:

- **Identificador "RS-NFXXX".** De las siglas *Requisito Software No Funcional*.
- **Descripción.** Definición del requisito.
- **Prioridad.** Determina el grado preferencia del requisito a ser aplicado.
- **Necesidad.** Determina el grado de importancia del requisito dentro del sistema.

RS-NF001			
Tipo	*Software*		
Nombre	*Python*.		
Descripción	Para la ejecución de la red de DQN se necesita la versión 2.7 de *Python*.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 30. RS-NF001

RS-NF002			
Tipo	*Software*		
Nombre	Librerías *Python* requeridas.		
Descripción	Para que pueda ejecutarse el subsistema *DQN-tensorflow* se necesitará instalar una serie de librerías: • *Tensorflow (versión 0.12.0)* • *Gym (versión 0.7.0)* • *atari-py (versión 0.0.21)* • *numpy, scipy, matplotlib, ipython, jupyter, pandas, sympy, nose* • *Pillow*		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 31. RS-NF002

RS-NF003			
Tipo	*Software*		
Nombre	Sistema operativo requerido.		
Descripción	El sistema solo podrá ejecutarse sobre Ubuntu en su versión 16.04 LTS. El buen funcionamiento del sistema para otras versiones de Ubuntu no está asegurado.		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 32. RS-NF003

RS-NF004			
Tipo	*Software*		
Nombre	Instalación de paquetes.		
Descripción	Para que pueda ejecutarse el subsistema *ALE-Atari-Width* se necesitará instalar una serie de paquetes: • *cmake* • *libsdl1.2-dev* • *libsdl-gfx1.2-dev* • *libsdl-image1.2-dev*		
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 33. RS-NF004

RS-NF005	
Tipo	*Hardware*
Nombre	Computadores permitidos.
Descripción	El sistema podrá ejecutarse en aquellos computadores que dispongan de los recursos mínimos para instalar el sistema operativo Ubuntu 16.04 LTS: • Procesador de 1 GHz. • 1.5 GB de memoria RAM. • 7 GB de almacenamiento interno libre. • Conector USB y acceso a internet.

RS-NF005			
Prioridad	☒ Alta ☐ Media ☐ Baja	Necesidad	☒ Alta ☐ Media ☐ Baja

Tabla 34. RS-NF005

6.1.4 Matriz de trazabilidad de requisitos de usuario y *software* funcionales

La Tabla 35 representa la matriz de trazabilidad entre los requisitos de usuario y los requisitos *software* funcionales:

		Requisitos de usuario							
		01	02	03	04	05	06	07	08
Requisitos *software* (funcionales)	01	X	X						
	02			X					
	03				X				
	04	X							
	05	X							
	06					X			
	07						X		
	08							X	
	09								X
	10								X
	11	X							
	12								X
	13								X
	14								X
	15	X							
	16								X

Tabla 35. Matriz de trazabilidad requisitos usuario – software funcionales

6.2 Diseño

En este apartado, se hablará del diseño del agente *CombinationAgent*. Se presentará a muy alto nivel el diseño de *ALE-Atari-Width* porque una parte importante del agente se implementará en dicho subsistema y será necesario comprender su funcionamiento y arquitectura. Para ello, se adjuntará un diagrama de las clases más relevantes del subsistema y de las nuevas clases diseñadas. También se especificarán los atributos y métodos de las nuevas clases que se diseñen y de las clases originales que precisen una modificación.

Por último, se incluirán diagramas de secuencia que describan el comportamiento interno del sistema en diversas situaciones.

6.2.1 Diagramas de clases

6.2.1.1 *ALE-Atari-Width*

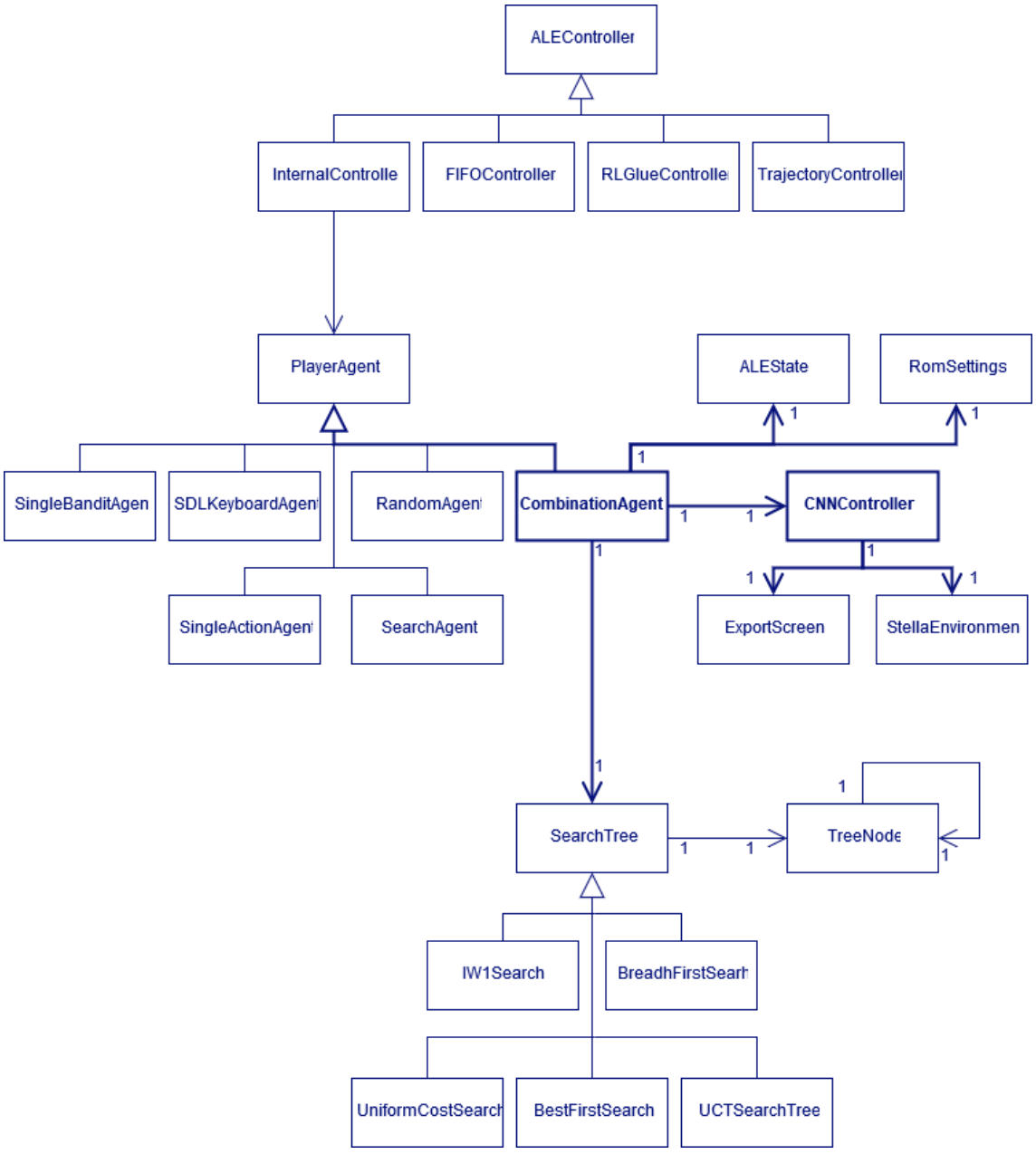


Ilustración 17. Diagrama de clases (en negrita, las nuevas clases y relaciones)

En el diagrama de la Ilustración 17 se observan 3 grupos de clases:

- **Clases herederas de *ALEController*.** Implementan diferentes tipos de controlador de la partida. Se usa fundamentalmente el controlador *InternalController*.
- **Clases heredadas de *PlayerAgent*.** Implementan diferentes tipos de agentes. El agente desarrollado por los autores del caso de estudio sobre IW [17] es *SearchAgent*.
- **Clases heredadas de *SearchTree*.** Implementan diferentes tipos de algoritmos de planificación (entre ellos IW, de la clase *IW1Search*), que serían utilizados por el agente *SearchAgent*. Para desarrollar el árbol de simulación, todos los algoritmos necesitan la implementación de "nodo" de la clase *TreeNode*.

Las nuevas clases planteadas son *CNNController* y *CombinationAgent* (en negrita en el diagrama). Se describen a continuación.

6.2.1.1.1 CNNController

Esta clase incorpora todas las herramientas necesarias para interactuar con la red implementada en el código *DQN-tensorflow*. Se centra en tratar las entradas de la red, enviarlas, llamar a la red, recibir las salidas y procesarlas.

La Ilustración 18 especifica los atributos y métodos de la clase:

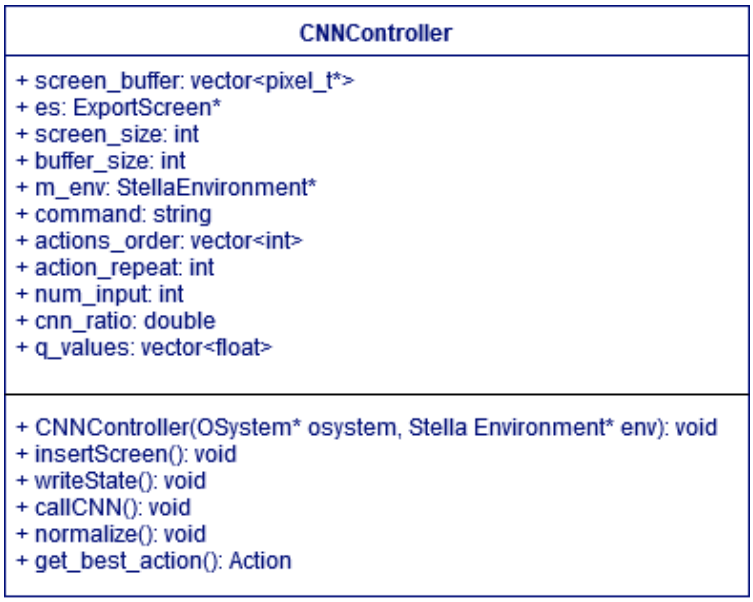


Ilustración 18. Especificación de la clase CNNController

En las siguientes tablas se describen los atributos, constructores y métodos de la clase:

Atributo	Tipo	Descripción
screen_buffer	*vector<pixel_t*>*	Historial de las imágenes más recientes de la partida. *pixel_t* es un tipo disponible en ALE para describir un pixel de la imagen.
es	*ExportScreen**	Contiene toda la lógica para transformar las imágenes del estado a formato RGB.
screen_size	*int*	Tamaño de la imagen (número de píxeles). Adopta como valor el producto de los parámetros de entrada del sistema *cnn_action_repeat* y *cnn_num_input*.
buffer_size	*int*	Tamaño máximo del *buffer.* Adopta el valor 160*210.
m_env	*StellaEnvironment*	Instancia del entorno de la partida. Dispone de herramientas para extraer las imágenes del estado actual de la partida. También es imprescindible para conocer los valores de los parámetros de entrada del sistema.
command	*string*	Cadena de caracteres que representa el comando para ejecutar el código *DQN-tensorflow*, que contiene la red. Adopta el valor: *"python ../DQN-tensorflow/main.py --env_name=<game> --mode=ale --use_gpu=False"*
actions_order	*vector<int>*	Describe a qué acción pertenece cada valor Q de salida de la red.
action_repeat	*int*	Adopta el valor del parámetro de entrada del sistema *cnn_action_repeat*.
num_input	*int*	Número de imágenes de entrada de la red de DQN. Adopta el valor del parámetro de entrada del sistema *cnn_action_repeat*.
cnn_ratio	*double*	Representa el peso de la red en la toma de decisiones del agente. El valor por defecto es el del parámetro de entrada *ratio*.
q_values	*vector<float>*	Listado de valores Q de salida de la red.

Tabla 36. Especificación de los atributos de CNNController

Constructor	
Entrada	*OSystem* osystem, StellaEnvironment* env*

Constructor	
Procesamiento	Inicializa los atributos de la clase, según lo mencionado en la Tabla 36. En el caso de *actions_order*, la inicialización es exclusiva de cada posible juego de Atari, porque cada juego cuenta con su propio conjunto mínimo de acciones.

Tabla 37. Especificación del constructor de CNNController

Métodos	
insertScreen()	
Entrada	*void*
Salida	*void*
Procesamiento	Obtiene una imagen asociada al estado actual de la partida, y la almacena en el *buffer* de imágenes.
writeState()	
Entrada	*void*
Salida	*void*
Procesamiento	Escribe en un fichero las imágenes (en formato RGB) más recientes del *buffer* de imágenes.
callCNN()	
Entrada	*void*
Salida	*void*
Procesamiento	Llama a *writeState()*, ejecuta la red de DQN invocando el código *DQN-tensorflow* y lee el fichero de salidas generado por la red.
normalize()	
Entrada	*void*
Salida	*void*
Procesamiento	Normaliza en un intervalo [0, 1] los valores Q leídos del fichero generado por la red.
get_best_action()	
Entrada	*void*
Salida	*Action*: Mejor acción considerada por la red de DQN.
Procesamiento	Averigua cuál es la mejor acción, buscando aquella con mayor valor Q asociado.

Tabla 38. Especificación de los métodos de CNNController

6.2.1.1.2 CombinationAgent

Esta clase implementa un nuevo agente, que combina DQN y los algoritmos de planificación. Por ello, la clase incluye una instancia del controlador de la red DQN y una instancia del algoritmo de planificación escogido con el parámetro *search_method* (IW, por ejemplo).

La clase contiene los mecanismos necesarios para combinar ambas técnicas durante una partida de un videojuego.

La Ilustración 19 especifica los atributos y métodos de la clase:

Ilustración 19. Especificación de la clase CombinationAgent

En las siguientes tablas se describen los atributos, constructores y métodos de la clase:

Atributo	Tipo	Descripción
m_curr_action	*Action*	Acción elegida en el *frame* actual. Se inicializa a UNDEFINED.
state	*ALEState*	Estado de la partida.
m_rom_settings	*RomSettings**	Instancia que aporta información sobre el estado de la partida.
sim_steps_per_node	*int*	Adopta el valor del parámetro de entrada del sistema *sim_steps_per_node*, que representa cada cuantos *frames* el agente decide una nueva acción.
search_method	*string*	Algoritmo de planificación definido para el agente. Adopta el valor del parámetro de entrada del sistema *search_method*.

Atributo	Tipo	Descripción
m_current_episode	*unsigned*	Episodio (partida) actual. Se inicializa a 0.
cnn	*CNNController**	Instancia del controlador de la red de DQN.
search_tree	*SearchTree**	Instancia del algoritmo de planificación.

Tabla 39. Especificación de los atributos de CombinationAgent

Constructor	
Entrada	*OSystem* _osystem, RomSettings* _settings, StellaEnvironment* _env, bool player_B*
Procesamiento	Inicializa alguno de los atributos de la clase, según lo establecido en la Tabla 39. Destacan las inicializaciones del controlador de la red de DQN (*cnn*) y del algoritmo de planificación (*search_tree*).

Tabla 40. Especificación del constructor de CombinationAgent

Métodos	
simulate()	
Entrada	*void*
Salida	*Action*: Mejor acción según el algoritmo de planificación.
Procesamiento	Se encarga de ejecutar el algoritmo de planificación, realizando una simulación de la partida desde el estado actual, de la misma manera que lo haría el agente *SearchAgent*. El resultado de la simulación es la acción elegida por el algoritmo de planificación.
modeConsensus()	
Entrada	*void*
Salida	*Action*: Acción consensuada por el algoritmo de planificación y por la red de DQN.
Procesamiento	Llama a *simulate()* y normaliza los refuerzos futuros asociados a cada acción. Llama a *callCNN*() y normaliza los valores Q de salida de la red. Realiza una suma ponderada del refuerzo futuro normalizado y del valor Q normalizado de cada acción. El factor de ponderación es el parámetro de entrada del sistema *ratio*.
modeProbability()	

Métodos	
Entrada	*void*
Salida	*Action*: Acción elegida por la red de DQN o por el algoritmo de planificación.
Procesamiento	Teniendo en cuenta el parámetro de entrada del sistema *ratio*, se elige aleatoriamente entre: **a)** Llamar a *simulate()* para obtener la mejor acción según el algoritmo de planificación, o **b)** llamar a *callCNN()* y a *get_best_action()* para obtener la mejor acción según la red de DQN.
act()	
Entrada	*void*
Salida	*Action*: Acción que se ejecutará para el *frame* actual de la partida.
Procesamiento	Escoge una acción en cada *frame* de la partida. Para ello, primero llama a *inserScreen()*. Después pueden darse las siguientes situaciones, según el *frame* de la partida: **a)** Que se llame a *modeConsensus()* o *modeProbability()* según lo indicado por el parámetro de entrada del sistema *mode*. **b)** Que se llame a *simulate()* según lo indicado por el parámetro de entrada del sistema *search_frequency*. **c)** Que se repita la acción escogida en el *frame* anterior, según lo indicado por el parámetro de entrada del sistema *sim_steps_per_node*.
get_best_action()	
Entrada	*vector<float> lista*: Lista de valores asociados a las acciones posibles. *vector<int> order*: Define a qué acción pertenece cada valor del parámetro *lista*.
Salida	*Action*: Mejor acción.
Procesamiento	Encuentra la mejor acción a partir de una lista de valores asociados a cada acción. La mejor acción es la de mayor valor.
episode_start()	
Entrada	*void*
Salida	*Action*: Acción que se ejecutará para el primer *frame* de la partida.

Métodos	
Procesamiento	Método heredado de *PlayerAgent.* Se encarga de elegir la primera acción de la partida.
episode_end()	
Entrada	*void*
Salida	*void*
Procesamiento	Método heredado de *PlayerAgent*. Se encarga de poner fin a la partida.
agent_step()	
Entrada	*void*
Salida	*Action*: Acción que se ejecutará para el *frame* actual de la partida.
Procesamiento	Método heredado de *PlayerAgent*. Se encarga de llamar *act()* para que el agente elija una acción en un *frame* determinado de la partida.

Tabla 41. Especificación de los métodos de CombinationAgent

6.2.1.1.3 SearchTree

Se han añadido dos métodos nuevos en *SearchTree* para facilitar a *CombinationAgent* obtener y tratar los refuerzos futuros derivados de la simulación. *SearchTree* implementa un algoritmo de planificación genérico que construye un árbol de simulación de la partida.

La Ilustración 20 especifica los nuevos métodos de la clase:

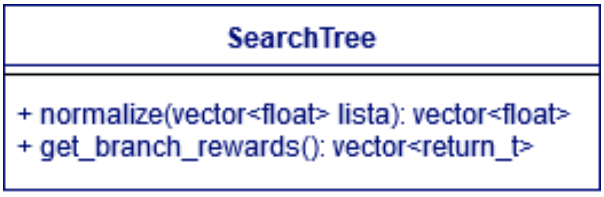


Ilustración 20. Ampliación clase SearchTree

En la Tabla 42 se describen los nuevos métodos de la clase:

Métodos	
normalize()	
Entrada	*vector<float> lista*: Lista de refuerzos futuros asociados a cada acción.
Salida	*vector<float>*: Lista de refuerzos futuros normalizados.
Procesamiento	Normaliza la lista de valores de entrada en un intervalo [0, 1].
get_branch_rewards()	
Entrada	*void*
Salida	*vector<return_t>*: Lista de refuerzos futuros resultantes de la simulación.
Procesamiento	Obtiene los refuerzos futuros asociados a cada acción, que son el resultado de la simulación del algoritmo de planificación.

Tabla 42. Especificación de los métodos de SearchTree

6.2.1.2 *DQN-tensorflow*

En este código no se diseñarán nuevas clases. Sin embargo, sí se realizarán algunas modificaciones de las clases originales.

6.2.1.2.1 Agent

Se trata de la clase principal de *DQN-tensorflow*. Implementa el algoritmo DQN. Es necesario añadir métodos que permitan ejecutar la red desde *ALE-Atari-Width*.

La Ilustración 21 especifica los nuevos métodos de la clase:

Ilustración 21. Ampliación de la clase Agent

En la Tabla 43 se describen los nuevos métodos de la clase:

Métodos	
predict2()	
Entrada	*array imagen*: Conjunto de imágenes.
Salida	*array*: Lista de valores Q por cada acción permitida.
Procesamiento	Ejecuta la red de DQN, pasando como entrada unas imágenes y recibiendo como respuesta una lista de valores Q asociados a cada acción.
play2()	
Entrada	*void*
Salida	*void*
Procesamiento	Lee las imágenes escritas en un fichero, llama a *process_image()* por cada una de ellas y se pasan como entrada de la red llamando a *predict2()*. Las salidas se escriben en otro fichero.

Tabla 43. Especificación de los métodos de Agent

6.2.1.2.2 Environment

Esta clase contiene todo lo relacionado con el entorno sobre el agente ejecuta las acciones. Se modificará para añadir el procesamiento de imágenes de entrada de la red.

La Ilustración 22 especifica los nuevos métodos de la clase:

Ilustración 22. Ampliación de la clase Environment

En la Tabla 44 se describen los nuevos métodos de la clase:

Métodos	
process_image()	
Entrada	*array image*: Imagen RGB
Salida	*array*: Imagen procesada
Procesamiento	Procesa la imagen de entrada siguiendo lo acordado en el caso de estudio sobre DQN [14]. **1.** Transformación de RGB a escala de grises. **2.** Reducción de tamaño de 160x210 a 110x84 píxeles. **3.** Recorte del área de interés (110x84 a 84x84 píxeles). En cada videojuego, el recorte de la imagen debe ser el más apropiado y no tiene por qué coincidir con el recorte del resto de videojuegos.

Tabla 44. Especificación de los métodos de Environment

6.2.2 Diagramas de secuencia

En la Ilustración 23 se muestra el primer diagrama de secuencia. Se corresponde con una ejecución del controlador *InternalController* con el agente *SearchAgent*, que utiliza IW como algoritmo de planificación (*IW1Search*). Se trata de un diagrama de referencia, puesto que representa la ejecución del agente original, el del caso de estudio sobre IW [17].

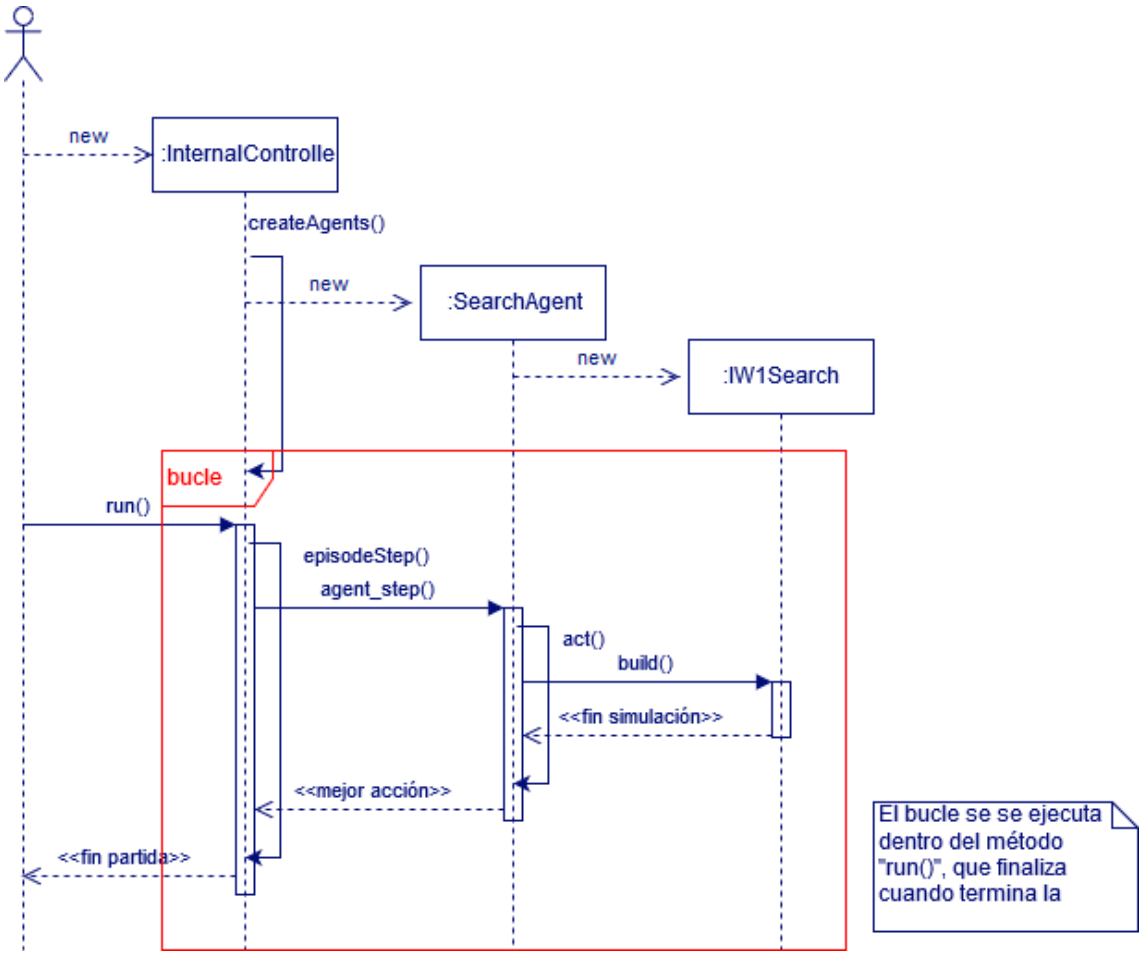


Ilustración 23. Diagrama de secuencia SearchAgent con IW

En la Ilustración 24 se muestra el segundo diagrama de secuencia. Se corresponde con una ejecución del controlador *InternalController* con el nuevo *CombinationAgent*. El agente utiliza IW como algoritmo de planificación (*IW1Search*) y el *modo* "consensus".

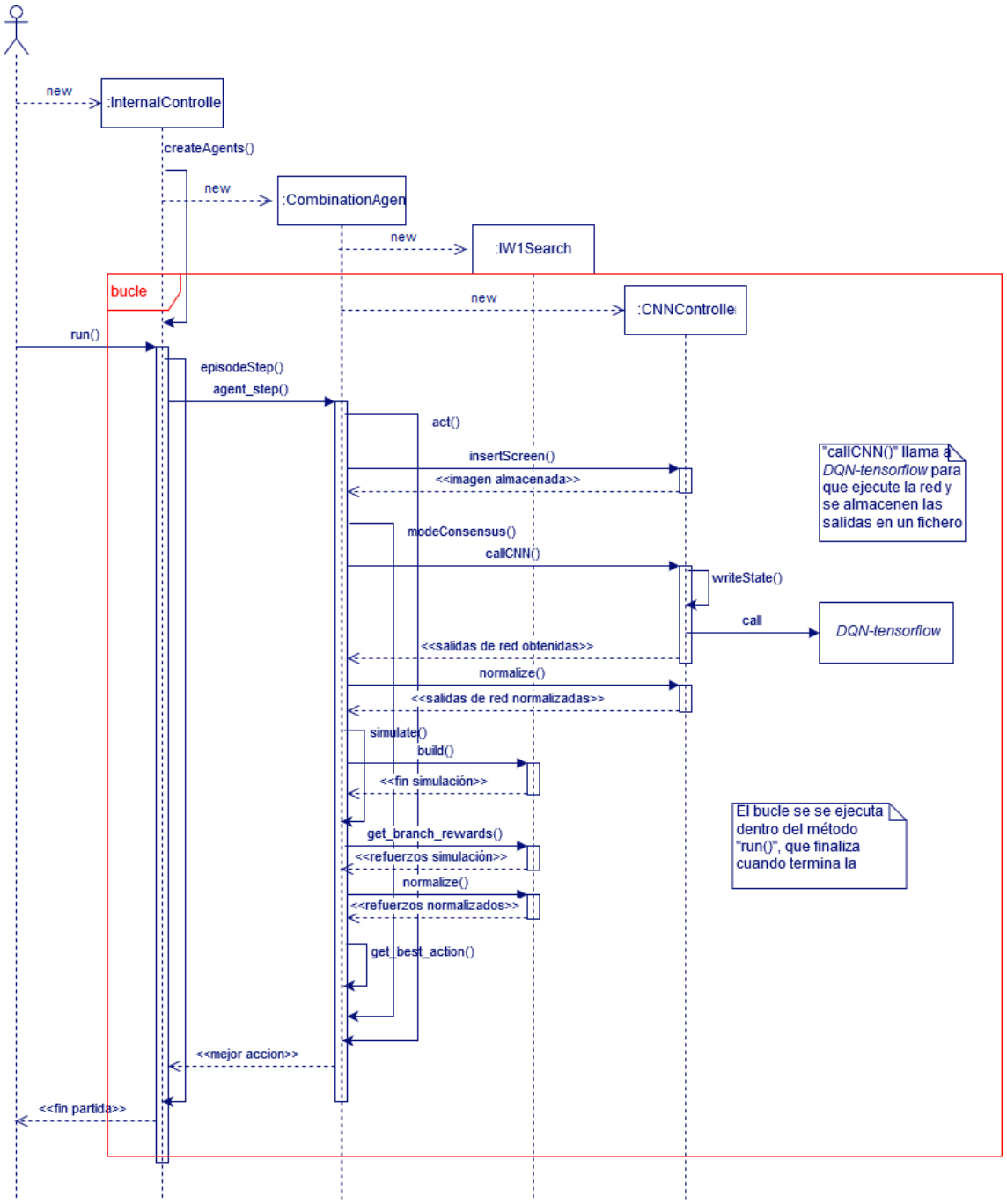


Ilustración 24. Diagrama de secuencia CombinationAgent con IW y modo Consensus

En la Ilustración 25 se muestra el tercer diagrama de secuencia. Se corresponde con una ejecución del controlador *InternalController* con el nuevo *CombinationAgent*. El agente utiliza IW como algoritmo de planificación (*IW1Search*) y el *modo* "probability".

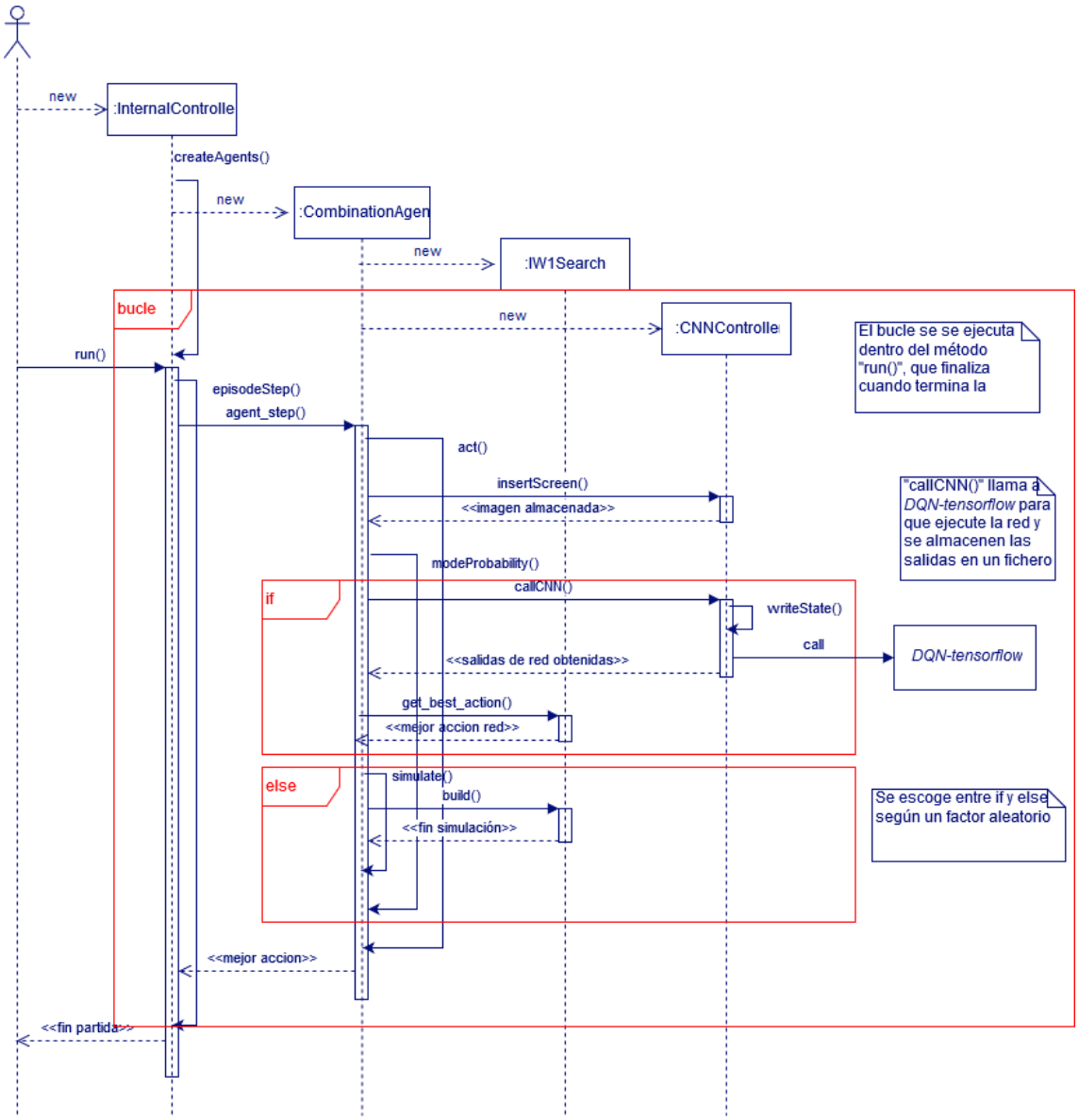


Ilustración 25. Diagrama de secuencia CombinationAgent con IW y modo Probability

En la Ilustración 26 se muestra el diagrama de secuencia que describe el comportamiento del subsistema *DQN-tensorflow* cuando es llamado desde el subsistema *ALE-Atari-Width*.

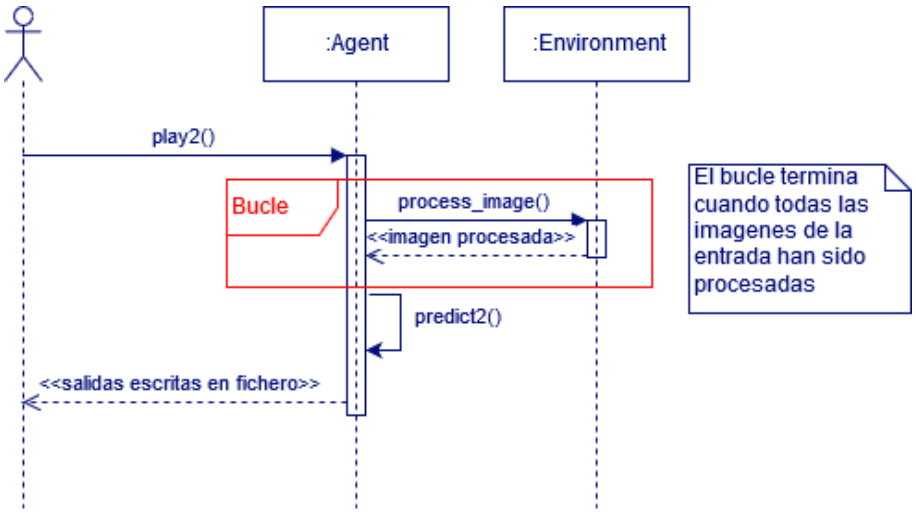


Ilustración 26. Diagrama de secuencia del subsistema DQN-tensorflow

6.3 Implementación

En esta etapa, se llevará a cabo la codificación del agente *CombinationAgent* y se integrará con los sistemas *software* ya existentes siguiendo el diseño expuesto en el apartado anterior.

Las únicas herramientas que se utilizarán para la codificación serán un editor de texto plano y la consola de Ubuntu 16.04.

En cuanto a los lenguajes de programación, se utilizará *C++* para implementar las clases *CNNController*, *CombinationAgent* y *SearchTree*, mientras que para las clases *Agent* y *Environment* se usará *Python*.

Las clases *CNNController* y *CombinationAgent* se encuentran en las carpetas *src/dqn* y *src/agents* del subsistema *ALE-Atari-Width*, respectivamente.

El sistema implementado está disponible en el repositorio *ALE-Atari-Combination* de *Github* [47].

6.4 Pruebas

6.4.1 Pruebas de sistema

Las pruebas de sistema se utilizarán para comprobar que todos los aspectos del sistema definidos en los requisitos *software* funcionales están correctamente implementados, y funcionan como se espera.

A continuación, se adjuntan las tablas de pruebas del sistema, que cuentan con los siguientes campos:

- **Identificador "PS-XXX".** De las siglas *Prueba del Sistema*.
- **Objetivo.** Propósito de la prueba.

- **Procedimiento de prueba.** Pautas que seguir para llevar a cabo la prueba.
- **Criterios de aceptación.** Condiciones que deben darse para que la prueba sea aprobada.
- **Requisitos relacionados.** Requisitos *software* comprobados por la prueba.

PS-001	
Objetivo	Comprobar que el sistema no da error cuando se escoge el agente *CombinationAgent* y el algoritmo de planificación IW.
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = "internal" • *player_agent* = "combination_agent" • *search_method* = "iw1" • *game* = {"Breakout-v0", "SpaceInvaders-v0"}
Criterios de aceptación	No se observan errores que indiquen error de implementación o incompatibilidad del sistema con el nuevo agente *CombinationAgent.*
Requisitos relacionados	RS-F001, RS-F003, RS-F004, RS-F011

Tabla 45. PS-001

PS-002	
Objetivo	Comprobar que el agente *CombinationAgent* ejecuta correctamente el modo "consensus".
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = "internal" • *player_agent* = "combination_agent" • *search_method* = "iw1" • *mode* = "consensus" • *ratio*, cualquier valor comprendido entre 0 y 1. Monitorizar en cada *frame* los siguientes datos: • La lista de refuerzos futuros de la simulación (normalizados y sin normalizar). • La lista de valores Q de salida de la red (normalizados y sin normalizar). • Suma ponderada de ambas listas.

PS-002	
	• Acción definitiva consensuada.
Criterios de aceptación	La partida se desarrolla con normalidad. No se aprecian errores, y los datos monitorizados indican que el procesamiento del modo “consensus” es correcto.
Requisitos relacionados	RS-F002, RS-F006, RS-F007

Tabla 46. PS-002

PS-003	
Objetivo	Comprobar que el agente *CombinationAgent* ejecuta correctamente el modo “probability”.
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = “internal” • *player_agent* = “combination_agent” • *search_method* = “iw1” • *mode* = “probability” • *ratio*, cualquier valor comprendido entre 0 y 1. Monitorizar en cada *frame* los siguientes datos: • Técnica escogida para ese *frame*. • La lista de refuerzos futuros de la simulación (sin normalizar), si la técnica escogida en el *frame* es la red de DQN. • La lista de valores Q de salida de la red (sin normalizar), si la técnica escogida en el *frame* es IW. • Acción definitiva escogida.
Criterios de aceptación	La partida se desarrolla con normalidad. No se aprecian errores, y los datos monitorizados indican que el procesamiento del modo “probability” es correcto.
Requisitos relacionados	RS-F002, RS-F006, RS-F008

Tabla 47. PS-003

PS-004	
Objetivo	Comprobar que el agente *CombinationAgent* decide una nueva acción según el número de *frames* marcado por el parámetro de entrada *sim_steps_per_node*.
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = "internal" • *player_agent* = "combination_agent" • *search_method* = "iw1" • *sim_steps_per_node* = 5 Monitorizar en cada *frame* los siguientes datos: • Acción escogida para ese *frame*.
Criterios de aceptación	Los datos monitorizados indican que, cada 5 *frames* de la partida, el agente escoge una nueva acción y la repite en los 4 *frames* siguientes.
Requisitos relacionados	RS-F005

Tabla 48. PS-004

PS-005	
Objetivo	Comprobar que el agente *CombinationAgent* realiza correctamente la interacción con la red de DQN.
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = "internal" • *player_agent* = "combination_agent" • *search_method* = "iw1" • *cnn_num_input* = 4 • *cnn_action_repeat* = 4 Monitorizar en cada *frame* los siguientes datos: • Estado del *buffer* de imágenes (tamaño actual, tamaño máximo, imágenes insertada y eliminada). • Imágenes del *buffer* escogidas para la entrada de la red, y número de imágenes. Observar en cada *frame* de la partida: • Que el fichero en el que se han escrito las imágenes existe y contiene las imágenes.

PS-005	
	• Que el fichero en el que se han escrito los valores Q de salida de la red existe y contiene dicha información.
Criterios de aceptación	Los datos monitorizados y la lectura de los ficheros indican que la interacción del agente con la red de DQN es correcta.
Requisitos relacionados	RS-F009, RS-F010, RS-F014, RS-F012, RS-F013

Tabla 49. PS-005

PS-006	
Objetivo	Comprobar que el agente *CombinationAgent* utiliza únicamente IW en un intervalo marcado por el parámetro de entrada *search_frequency*.
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = "internal" • *player_agent* = "combination_agent" • *search_method* = "iw1" • *search_frequency* = 5 Monitorizar en cada *frame* los siguientes datos: • Quién toma la decisión en ese *frame* (*modo* o IW). • Acción escogida para ese *frame*.
Criterios de aceptación	Los datos monitorizados indican que, cada 5 decisiones tomadas por el agente, una es el resultado de la ejecución exclusiva de IW. Las 4 decisiones restantes son el resultado de ejecutar el *modo* elegido.
Requisitos relacionados	RS-F015

Tabla 50. PS-006

PS-007	
Objetivo	Comprobar el correcto procesamiento de las imágenes escritas en el fichero.
Procedimiento de prueba	Ejecutar el sistema teniendo en cuenta que los siguientes parámetros deben adoptar los siguientes valores: • *game_controller* = "internal" • *player_agent* = "combination_agent"

PS-007	
	• *search_method* = “iw1” Mostrar en cada *frame* de la partida las imágenes originales y las procesadas.
Criterios de aceptación	Tanto las imágenes originales como las procesadas son las esperadas. Las originales tienen dimensión 160x210. Las procesadas han sido escaladas, recortadas y transformadas escala de grises, y su dimensión es 84x84.
Requisitos relacionados	RS-F016

Tabla 51. PS-007

La Tabla 52 representa la matriz de trazabilidad entre los requisitos *software* funcionales y las pruebas del sistema:

		Pruebas del sistema						
		01	02	03	04	05	06	07
Requisitos *software* (funcionales)	01	X						
	02		X	X				
	03	X						
	04	X						
	05				X			
	06		X	X				
	07		X					
	08			X				
	09					X		
	10					X		
	11	X						
	12					X		
	13					X		
	14					X		
	15						X	
	16							X

Tabla 52. Matriz de trazabilidad pruebas del sistema – requisitos software funcionales

6.4.2 Pruebas unitarias

Las pruebas unitarias se utilizarán para comprobar que las unidades del sistema (clases y métodos diseñados) están correctamente implementados, y funcionan como se espera.

A continuación, se adjuntan las tablas de pruebas unitarias, que cuentan con los siguientes campos:

- **Identificador "PU-XXX".** De las siglas *Prueba Unitarias*.
- **Subsistema.** Nombre del subsistema al que pertenece el método.
- **Clase.** Nombre de la clase a la que pertenece el método.
- **Método.** Nombre del método.
- **Criterios de aceptación.** Condiciones que deben darse para que la prueba sea aprobada.

PU-001	
Subsistema	*ALE-Atari-Width*
Clase	*CNNController*
Método	*insertScreen()*
Criterios de aceptación	Al finalizar la ejecución del método, se debe haber insertado la imagen correspondiente al *frame* actual al final del *buffer* de imágenes. En ningún caso, el *buffer* debe superar el límite de tamaño.

Tabla 53. PU-001

PU-002	
Subsistema	*ALE-Atari-Width*
Clase	*CNNController*
Método	*writeState()*
Criterios de aceptación	Al finalizar la ejecución del método, debe haberse creado o sobrescrito un fichero que contenga las imágenes escogidas del *buffer* de imágenes. El número de imágenes debe coincidir con el valor del parámetro de entrada del sistema *cnn_num_imput,* y deben estar ordenadas de más antiguas a más nuevas. Las imágenes elegidas deben tener en cuenta el parámetro de entrada del sistema *cnn_action_repeat*.

Tabla 54. PU-002

PU-003	
Subsistema	*ALE-Atari-Width*
Clase	*CNNController*
Método	*callCNN()*
Criterios de aceptación	Al finalizar la ejecución del método, debe haberse creado o sobrescrito un fichero que contenga los valores Q de salida de la red. El número de valores debe coincidir con el número de acciones permitidas por el videojuego escogido.

Tabla 55. PU-003

PU-004	
Subsistema	*ALE-Atari-Width*
Clase	*CNNController*
Método	*normalize()*
Criterios de aceptación	Al finalizar la ejecución del método, los últimos valores Q obtenidos por la red estarán normalizados en un intervalo [0, 1].

Tabla 56. PU-004

PU-005	
Subsistema	*ALE-Atari-Width*
Clase	*CNNController*
Método	*get_best_action()*
Criterios de aceptación	Dados los últimos valores Q obtenidos por la red, el método devuelve la acción con mayor valor Q asociado.

Tabla 57. PU-005

PU-006	
Subsistema	*ALE-Atari-Width*
Clase	*CombinationAgent*
Método	*simulate()*
Criterios de aceptación	Al finalizar la ejecución del método, se ha realizado la simulación y se ha construido un árbol con la información sobre los refuerzos futuros de cada acción posible.

Tabla 58. PU-006

PU-007	
Subsistema	*ALE-Atari-Width*
Clase	*CombinationAgent*
Método	*modeConsensus()*
Criterios de aceptación	Se devuelve la acción consensuada por el algoritmo de planificación y la red de DQN.

Tabla 59. PU-007

PU-008	
Subsistema	*ALE-Atari-Width*
Clase	*CombinationAgent*
Método	*modeProbability()*
Criterios de aceptación	Se devuelve la acción escogida por el algoritmo de planificación o por la red de DQN.

Tabla 60. PU-008

PU-009	
Subsistema	*ALE-Atari-Width*
Clase	*CombinationAgent*
Método	*act()*
Criterios de aceptación	La acción devuelta se corresponde con la obtenida por *modeConsensus()*, *modeProbability()* o *simulate()*, según el *frame* de la partida.

Tabla 61. PU-009

PU-010	
Subsistema	*ALE-Atari-Width*
Clase	*CombinationAgent*
Método	*get_best_action()*
Criterios de aceptación	Dada la lista de valores numéricos de entrada, el método devuelve la acción con mayor valor asociado.

Tabla 62. PU-010

PU-011	
Subsistema	*ALE-Atari-Width*
Clase	*SearchTree*
Método	*normalize()*
Criterios de aceptación	Al finalizar la ejecución del método, los últimos refuerzos futuros obtenidos por la simulación estarán normalizados en un intervalo [0, 1].

Tabla 63. PU-011

PU-012	
Subsistema	*ALE-Atari-Width*
Clase	*SearchTree*
Método	*get_branch_rewards()*
Criterios de aceptación	Se devuelve una lista de valores numéricos correspondientes con los refuerzos futuros obtenidos con la última simulación del algoritmo de planificación.

Tabla 64. PU-012

PU-013	
Subsistema	*DQN-tensorflow*
Clase	*Agent*
Método	*predict2()*
Criterios de aceptación	Dada la entrada de la red (imágenes procesadas de dimensión 84x84), se obtiene una lista de valores numéricos reales que representa los valores Q asociados a cada acción. El número de

PU-013	
	valores coincide con el número de posibles acciones para el videojuego elegido.

Tabla 65. PU-013

PU-014	
Subsistema	*DQN-tensorflow*
Clase	*Agent*
Método	*play2()*
Criterios de aceptación	Dado un fichero de imágenes, al finalizar la ejecución del método debe haberse creado o sobrescrito un fichero que contenga los valores Q de salida de la red. El número de valores debe coincidir con el número de acciones permitidas por el videojuego escogido.

Tabla 66. PU-014

PU-015	
Subsistema	*DQN-tensorflow*
Clase	*Environment*
Método	*process_image()*
Criterios de aceptación	El array de salida representa la imagen RGB original, pero reducida, recortada y transformada a escala de grises. La debe ser 84x84 píxeles.

Tabla 67. PU-015

6.4.3 Pruebas de implantación

Las pruebas de implantación se utilizarán para comprobar que el sistema se instala correctamente según lo establecido en los requisitos *software* no funcionales.

A continuación, se adjuntan las tablas de pruebas de implantación, que cuentan con los siguientes campos:

- **Identificador "PI-XXX".** De las siglas *Prueba de Implantación*.
- **Objetivo.** Objetivo de la prueba.
- **Procedimiento de prueba.** Pautas que seguir para llevar a cabo la prueba.
- **Criterios de aceptación.** Condiciones que deben darse para que la prueba sea aprobada.

PI-001	
Objetivo	Ejecutar satisfactoriamente el sistema en un computador con sistema operativo Ubuntu 16.04.
Procedimientos de prueba	Disponer de Ubuntu 16.04 y de *Python 2.7*. Instalar una serie de librerías de *Python*. Los comandos: • *sudo pip install tensorflow==0.12.0* • *sudo pip install gym==0.7.0* • *pip install atari-py==0.0.21* • *python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose* • *sudo pip install Pillow* Instalar una serie de paquetes de Ubuntu. Los comandos: • *sudo apt-get install cmake* • *sudo apt-get install libsdl1.2-dev* • *sudo apt-get install libsdl-gfx1.2-dev* • *sudo apt-get install libsdl-image1.2-dev*
Criterios de aceptación	El sistema no produce ningún error en su ejecución.

Tabla 68. PI-001

6.5 Verificación

Como el usuario final es el propio desarrollador del *software*, al llevar a cabo las pruebas de la etapa anterior, queda verificado el buen funcionamiento del producto.

6.6 Mantenimiento

El mantenimiento será **preventivo**, lo que implica que se revisará habitualmente con el objetivo de detectar errores. También será un mantenimiento **perfectivo**, ya que la experimentación que se realizará con el *software* puede revelar la necesidad de nuevas funcionalidades.

De acuerdo con las nuevas decisiones tomadas durante la experimentación, se han implementado nuevas características dentro de la etapa de mantenimiento:

- En la etapa 3 de la experimentación se ha decidido crear el parámetro *search_frequency*, y se ha incluido como requisito *software* funcional RS-F015.
- En la etapa 1 de la experimentación se ha decidido cambiar el procesamiento de imágenes, y se ha incluido como requisito *software* funcional RS-F016.

7 Evaluación

7.1 Metodología de experimentación

Una vez construido el sistema *software*, se evaluará por medio de una experimentación, que estará dividida en 3 etapas. Los resultados de cada etapa servirán para tomar decisiones sobre las etapas posteriores.

Se explican a continuación:

- **Etapa 1.**

  Se realizará una experimentación con DQN para poner a punto el algoritmo, de cara a la etapa siguiente. Partiendo de un videojuego de referencia y de una configuración de parámetros inicial, se propondrán posibles mejoras y se probarán. Estas mejoras pueden ser cambios de valores de los parámetros de DQN o ligeros cambios en la implementación del algoritmo. Con cada experimento se incluirán las gráficas de evolución del aprendizaje de la red, que servirán para determinar la calidad del entrenamiento.

- **Etapa 2.**

  Con DQN configurado en la etapa anterior, ahora se propone utilizar el algoritmo para entrenar las redes convolucionales de un subconjunto de videojuegos de Atari. Al igual que en la etapa 1, por cada juego, se adjuntarán diversas gráficas de evolución del aprendizaje de la red.

- **Etapa 3.**

  Una vez que las redes hayan sido entrenadas, se procederá a la experimentación con los agentes. Esta etapa es muy exhaustiva y consta de 2 fases. En primer lugar, se ejecutará el agente *SearchAgent* utilizando IW como algoritmo de planificación. Después, se experimentará con el nuevo agente *CombinationAgent*, utilizando también IW como algoritmo de planificación. Los resultados de ambas fases serán comparados con el fin de determinar si la combinación IW+DQN es más efectiva que IW individual.

En la Ilustración 27 se resume la metodología:

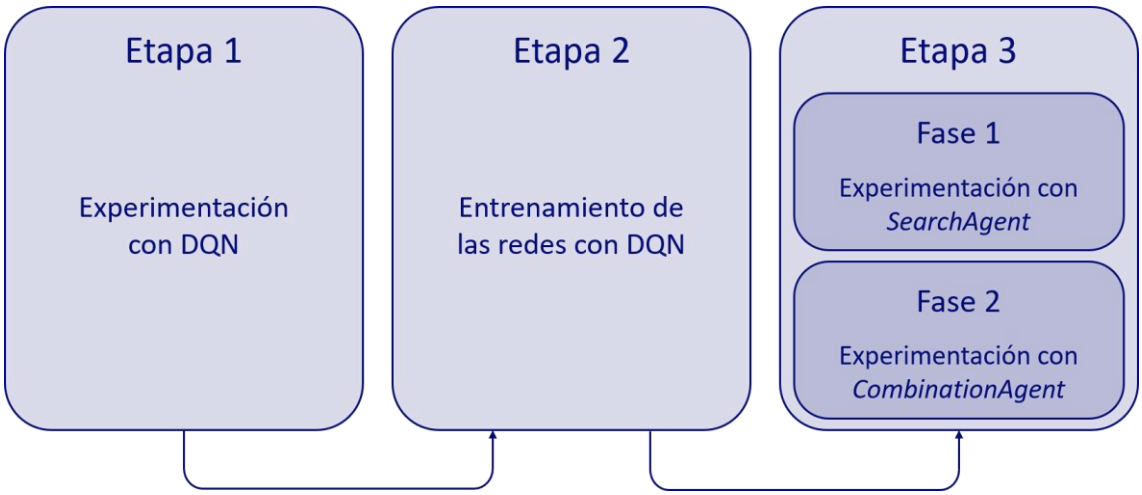


Ilustración 27. Diagrama de metodología de experimentación

7.2 Parámetros de la experimentación

La Tabla 69 incluye los parámetros con los que se experimentará en las etapas 1 y 2:

Parámetro	Dominio	Descripción
memory_size	$x \in \mathbb{N} \mid x > 0$	Cantidad máxima de experiencias que puede almacenar la *memory replay* de DQN.
batch_size	$x \in \mathbb{N} \mid x > 0$	Número de experiencias de la *memory replay* que se usan para entrenar la red convolucional.
discount	$x \in \mathbb{R} \mid 0 \leq x \leq 1$	Factor de descuento.
learning_rate	$x \in \mathbb{R} \mid 0 \leq x \leq 1$	Velocidad de aprendizaje.
history_length	$x \in \mathbb{N} \mid x > 0$	Número de *frames* más recientes, que se pasan como entrada de la red convolucional.
epsilon_start	$x \in \mathbb{R} \mid 0 \leq x \leq 1$	Valor inicial de ϵ de la estrategia ϵ-greedy.
epsilon_end	$x \in \mathbb{R} \mid 0 \leq x \leq 1$	Valor final de ϵ de la estrategia ϵ-greedy.
epsilon_end_t	$x \in \mathbb{N} \mid x > 0$	Número de iteraciones necesarias para que ϵ alcance su valor mínimo *epsilon_end*.
momentum	$x \in \mathbb{R} \mid 0 \leq x \leq 1$	*Momentum* del algoritmo de optimización por descenso de gradiente utilizado por DQN.
action_repeat	$x \in \mathbb{N} \mid x > 0$	Número de veces que se repite la misma acción elegida.

Tabla 69. Descripción de parámetros de las etapas 1 y 2

La Tabla 70 incluye los parámetros con los que se experimentará en la etapa 3:

Parámetro	Dominio	Descripción
juego	Títulos de juegos de *Atari 2600*	Título del juego de *Atari 2600*.
ratio	$x \in \mathbb{R} \mid 0 \leq x \leq 1$	Porcentaje de influencia de la red de DQN en las decisiones de IW, durante la partida.
limite_simulacion	$x \in \mathbb{N} \mid x > 0$	Límite de tamaño del árbol de simulación de IW.
modo	$x \in \{Consensus, Probability\}$	Criterio del agente *CombinationAgent* para elegir la mejor acción, dado un estado de la partida: **a)** ***Consensus.*** La mejor acción a^* es aquella con mayor refuerzo normalizado total $R^N(a^*)$, que se obtiene de la suma ponderada de los refuerzos normalizados aportados por la simulación de IW, $R^N_{IW}(a^*)$, y por la red de DQN, $R^N_{DQN}(a^*)$. $R^N(a) = ratio * R^N_{DQN}(a) + (1 - ratio) * R^N_{IW}(a), \forall\, a \in A$ $a^* = \arg\max_a R^N(a), \forall\, a \in A$ **b)** ***Probability.*** La mejor acción a^* es la propuesta por la red de DQN con probabilidad $ratio$, o la propuesta por IW con probabilidad $(1 - ratio)$.

Tabla 70. Descripción de parámetros de la etapa 3

7.3 Experimentación

7.3.1 Etapa 1: Experimentación con DQN

En esta etapa se llevarán a cabo una serie de experimentos para elegir la mejor configuración de parámetros de DQN, y para mejorar algunos aspectos de la implementación [21].

La configuración base de parámetros ha sido extraída del artículo de tipo experimental de *DeepMind* [16], porque ha sido la que mejor les ha funcionado. Según los autores, los valores de los parámetros se definieron por medio de una experimentación informal sobre un pequeño subconjunto de juegos de Atari (*Breakout*, *Pong*, *Seaquest*, *Space Invaders* y *Beam Rider*). La configuración de parámetros obtenida de la experimentación fue posteriormente utilizada para entrenar el resto de los videojuegos.

En la Tabla 71 se adjuntan los valores de los parámetros:

Parámetro	Valor
memory_size	300000
batch_size	32
discount	0,995
learning_rate	0,00025
history_length	4
epsilon_start	1
epsilon_end	0,1
epsilon_end_t	1000000
momentum	0,95

Tabla 71. Configuración base de la etapa 1

El único parámetro que ha sufrido una variación es *memory_size*, que en el artículo mencionado adopta el valor 1000000. La reducción se debe a las limitaciones de memoria RAM del computador utilizado para la experimentación. *memory_size* hace referencia al límite de experiencias que pueden almacenarse durante el entrenamiento de DQN, y un valor 1000000 sobrepasa el límite permitido por una memoria RAM de 8GB.

Se ha tenido en cuenta otra limitación, en este caso temporal. Para que el entrenamiento de una red convolucional con DQN en el ámbito de los juegos de Atari sea eficiente en tiempo es conveniente utilizar una GPU, puesto que el algoritmo de aprendizaje de las redes convolucionales puede exprimir la alta capacidad de paralelización de un procesador gráfico. Sin embargo, no se dispone de un computador con una GPU compatible, por lo que el procesamiento se lleva a cabo necesariamente con una CPU. Esto significa que el aprendizaje será sensiblemente más lento, y por ello ha sido necesario establecer un límite de tiempo. Cada ejecución de DQN tendrá un límite de **3 millones de iteraciones**, que se corresponde con unas 20 horas de entrenamiento ininterrumpido, aproximadamente.

El videojuego de referencia utilizado es *Breakout* porque también fue usado en el artículo de referencia [16] para experimentar con los parámetros de DQN. No se considera necesario utilizar más juegos por el coste temporal que ello conllevaría.

Por cada experimento, se incluirán gráficas que evalúan la evolución del entrenamiento. Cada 25000 iteraciones se realizan tres evaluaciones:

- **Refuerzo medio (*avg_reward*).** Media de los refuerzos obtenidos en las últimas 25000 iteraciones del entrenamiento.

$$avg\_reward = \frac{\sum_{i=0}^{25000} r_i}{25000} \mid r_i, \text{ refuerzo obtenido en la iteración } i$$

- **Refuerzo medio por episodio (*avg_ep_reward*).** Media de refuerzos totales obtenidos en todos los episodios (partidas) ejecutados en las últimas 25000 iteraciones.

$$avg\_ep\_reward = \frac{\sum_{i=0}^{n} R_i}{n} \mid R_i, \text{ refuerzo total del episodio } i; n, \text{ número de episodios}$$

- **Refuerzo máximo por episodio (*max_ep_reward*).** Máximo refuerzo total obtenido en uno de los episodios (partidas) ejecutados en las últimas 25000 iteraciones.

$$max\_ep\_reward = \max_{0 \le i \le n} R_i \mid R_i, \text{ refuerzo total del episodio } i; n, \text{ número de episodios}$$

En los siguientes subapartados se detallan los experimentos.

7.3.1.1 Experimento 1

En este experimento se utiliza la implementación original de DQN [21], sin ninguna modificación y con la configuración de parámetros expuesta en la Tabla 71.

La duración del entrenamiento ha sido de **19 horas, 6 minutos y 56 segundos**. A continuación, se exponen las gráficas de evolución:

Ilustración 28. Evolución del avg_reward del experimento 1 (etapa 1)

Ilustración 29. Evolución del avg_ep_reward del experimento 1 (etapa 1)

Ilustración 30. Evolución del max_ep_reward del experimento 1 (etapa 1)

Las gráficas sugieren que la red ha aprendido. Sin embargo, el entrenamiento parece no ser tan efectivo como el ofrecido en el caso de estudio sobre DQN [16]. Esto puede deberse a que se necesite más tiempo de aprendizaje, o a que tuvo que reducirse el valor recomendado de *memory_size*. También puede deberse a que las entradas de la red son imágenes sin procesar, algo que no recomienda el caso de estudio de *DeepMind* [14]. El siguiente experimento tratará de solucionar esto último.

7.3.1.2 Experimento 2

En este experimento se mantiene la configuración de parámetros base de la Tabla 71, pero se modifica un detalle de la implementación de DQN. En el código original, la

entrada de la red convolucional es una secuencia de imágenes en bruto, transformadas a escala de grises. Lo que se propone en este experimento es procesarlas ligeramente para optimizar el aprendizaje de la red, tal y como se comenta en el caso de estudio sobre DQN [14].

[4x160x210] **ORIGINAL**

[4x84x84] **PROPUESTO**

Ilustración 31. Cambio propuesto en el experimento 2 de la etapa 1

Como puede observarse en la Ilustración 31, con el cambio propuesto desaparecen elementos de la imagen que no aportan nada al aprendizaje de la red, como la puntuación y el número de vidas del jugador.

La duración del entrenamiento ha sido de **17 horas, 17 minutos y 35 segundos**. A continuación, se exponen las gráficas de evolución:

Ilustración 32. Evolución del avg_reward del experimento 2 (etapa 1)

Ilustración 33. Evolución del avg_ep_reward del experimento 2 (etapa 1)

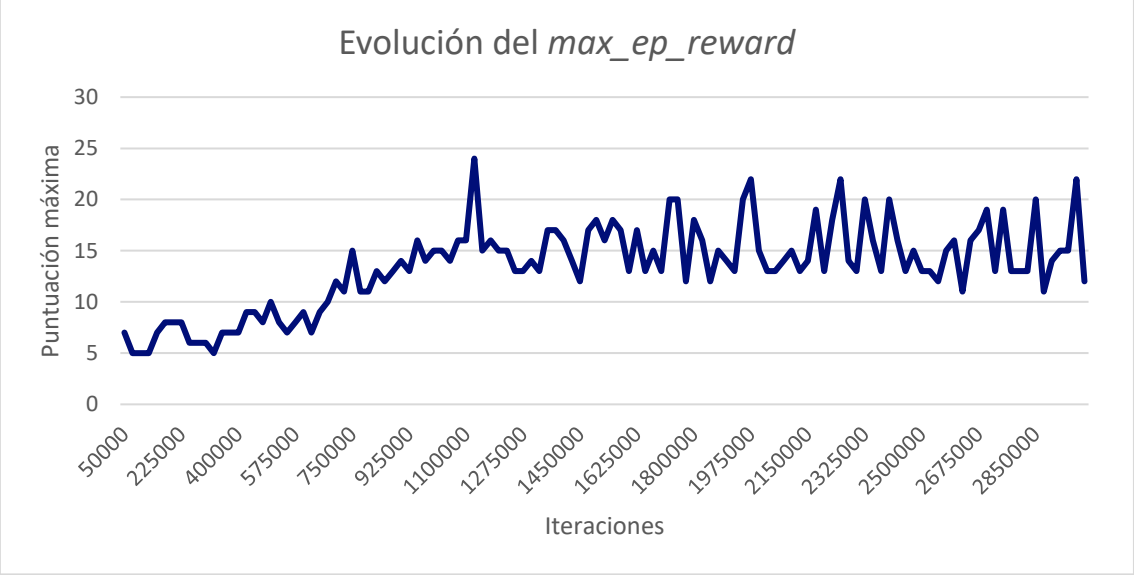


Ilustración 34. Evolución del max_ep_reward del experimento 2 (etapa 1)

Los resultados parecen ser ligeramente mejores en este experimento. Se observa en las gráficas de evolución de *avg_reward* y *avg_ep_reward*. Aunque la mejora no es muy significativa, se mantendrá el cambio propuesto para los siguientes experimentos porque es acorde a lo que establece el caso de estudio sobre DQN [14].

7.3.1.3 Experimento 3

En el experimento 2 se ha observado que el aprendizaje de la red sufre un estancamiento prematuro. Puede deberse a que a no se ha explorado lo suficiente en un entorno en el que el número de estados es muy alto.

Podría ser necesario aumentar el grado de exploración. Por ello, se ha decidido incrementar el valor del parámetro *epsilon_end_t*, que representa la iteración en la que ϵ alcanza su mínimo valor (véase la Tabla 72).

Parámetro	Valor original	Valor propuesto
epsilon_end_t	1000000	2000000

Tabla 72. Cambio propuesto en el experimento 3 de la etapa 1

La duración del entrenamiento ha sido de **18 horas, 32 minutos y 44 segundos**. A continuación, se exponen las gráficas de evolución:

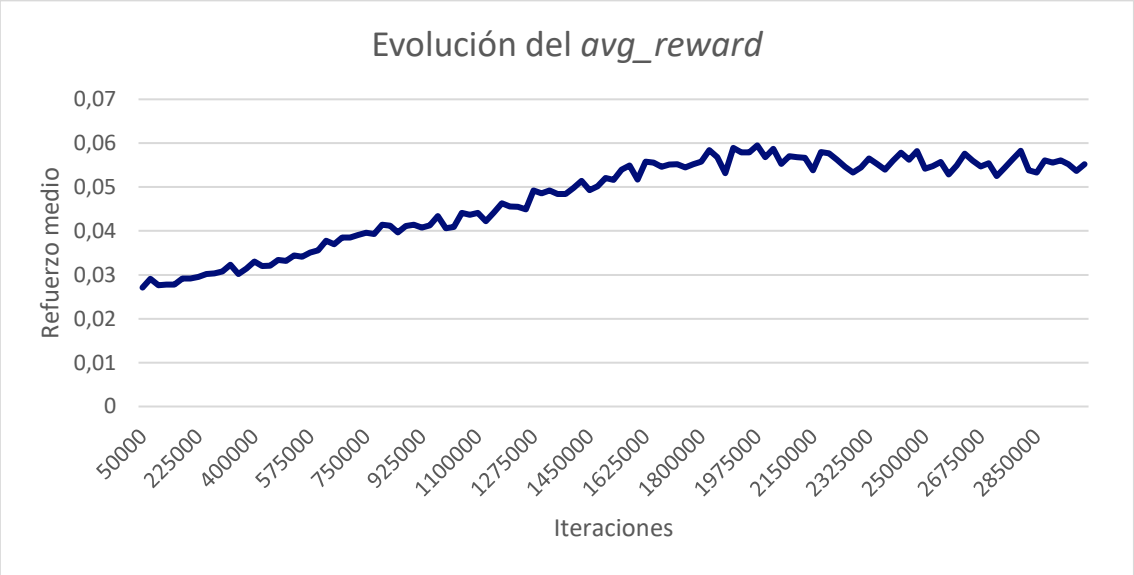


Ilustración 35. Evolución del avg_reward del experimento 3 (etapa 1)

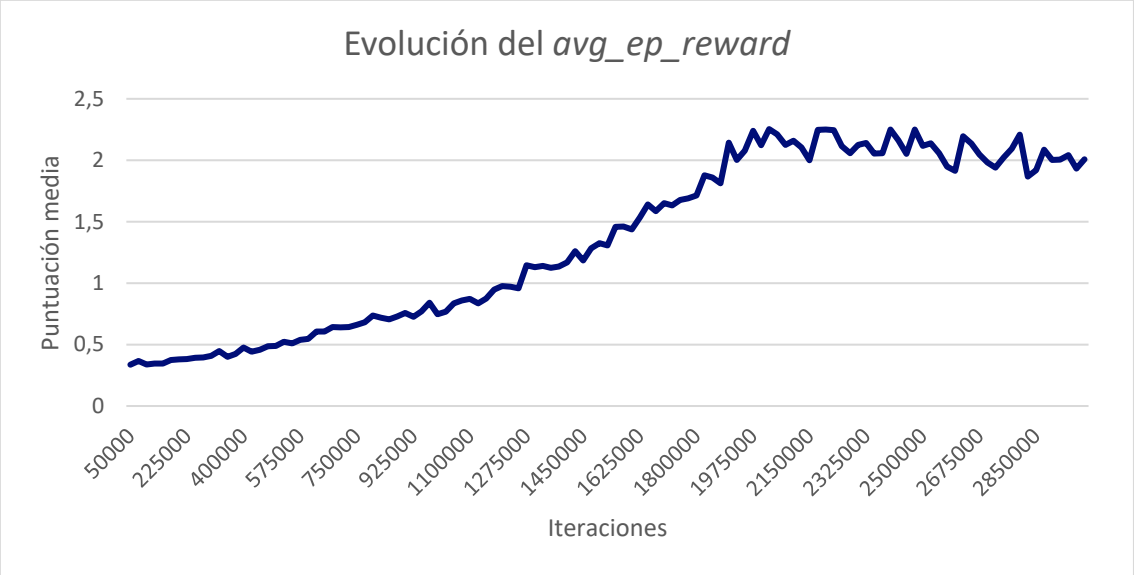


Ilustración 36. Evolución del avg_ep_reward del experimento 3 (etapa 1)

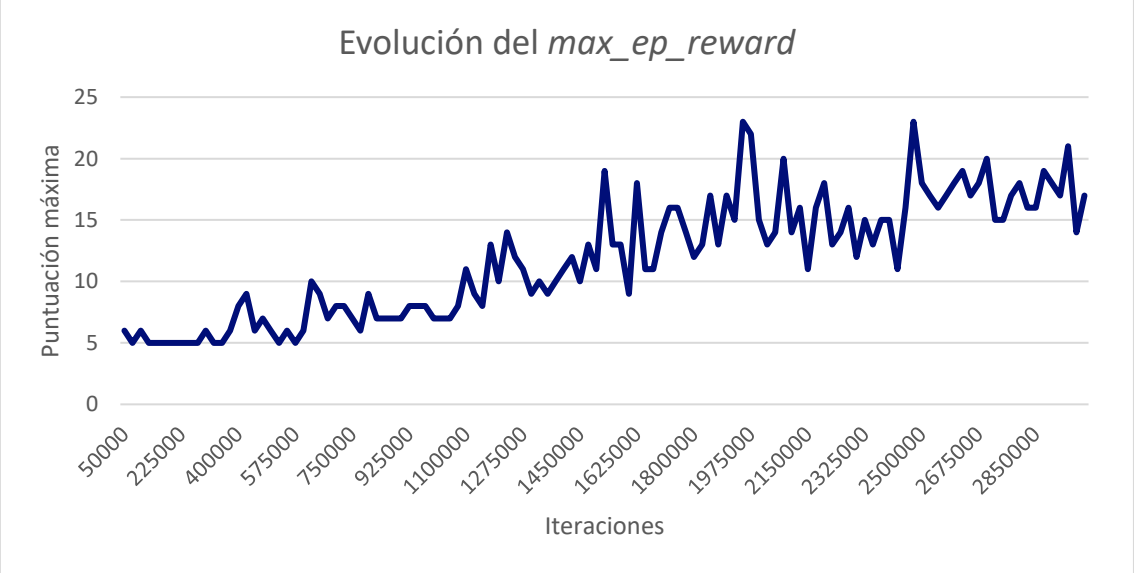


Ilustración 37. Evolución del max_ep_reward del experimento 3 (etapa 1)

Con el cambio propuesto, no se ha observado ninguna mejora en el entrenamiento. Las gráficas muestran que la explotación no ha sido más efectiva que en el experimento anterior. Por ello, se considera que el experimento 2 es mejor que este.

7.3.2 Etapa 2: Entrenamiento de las redes de DQN

Con la etapa 1 completada, se entrenarán las redes convolucionales de cada uno de los videojuegos de Atari propuestos. Para ello, se utilizará el código de DQN en las mismas condiciones que se indicaron en el experimento 2:

- Configuración de parámetros de la Tabla 71.

- Cambio en el procesado de las imágenes de entrada de la red.
- Límite de entrenamiento de 3 millones de iteraciones.

Los videojuegos con los que se experimentará deben cumplir alguna de las siguientes características:

a) Que sea resuelto por IW con una puntuación baja, en comparación con otras técnicas de *planning*, o

b) que sea resuelto por IW con una puntuación significativamente mayor que con una red convolucional entrenada con DQN, o

c) que sea resuelto por una red convolucional entrenada con DQN con una puntuación significativamente mayor que con IW.

Se busca ese perfil concreto de juegos porque interesa averiguar en la etapa 3 si la combinación IW+DQN mejora las puntuaciones de IW y DQN por separado, y las puntuaciones de otras técnicas de *planning*.

En los artículos sobre DQN [16] e IW [17] estudiados en el estado del arte se encuentra la información relativa a las puntuaciones de los juegos de Atari utilizando DQN y técnicas de *planning* (IW incluido), respectivamente.

7.3.2.1 *Breakout*

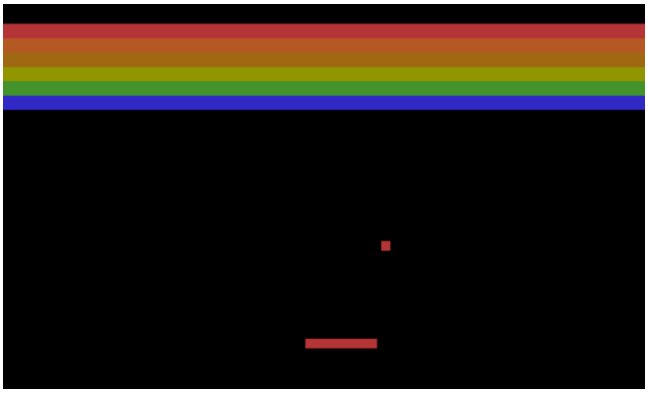


Ilustración 38. Captura de Breakout (obtenida de 4kepics [48])

Breakout (véase la Ilustración 38) es un juego de mecánica muy sencilla que cuenta con un conjunto de 6 acciones posibles. El juego consiste en destruir todos los bloques de colores utilizando una bola que no debe tocar el suelo más de 5 veces a lo largo de la partida. Los niveles de bloques superiores (colores cálidos) otorgan al jugador mayor puntuación que los niveles inferiores (colores fríos).

Se ha escogido este juego porque, además de ser utilizado como referencia durante la etapa 1 de la experimentación, cumple con dos criterios deseados: (1) IW lo resuelve

peor que una red entrenada con DQN (384 puntos frente a 401.2±26.9 puntos), y (2) IW es superado por otras técnicas de *planning* expuestas en el caso de estudio sobre IW [17].

La duración del entrenamiento ha sido de **17 horas, 17 minutos y 35 segundos**. A continuación, se exponen las gráficas de evolución:

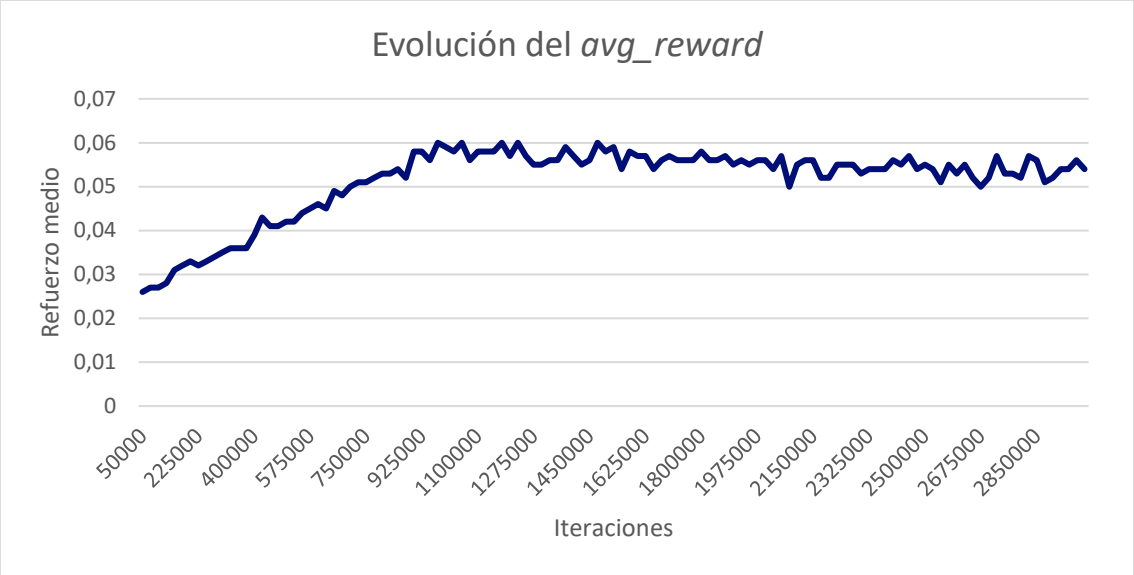


Ilustración 39. Evolución del avg_reward para Breakout (etapa 2)

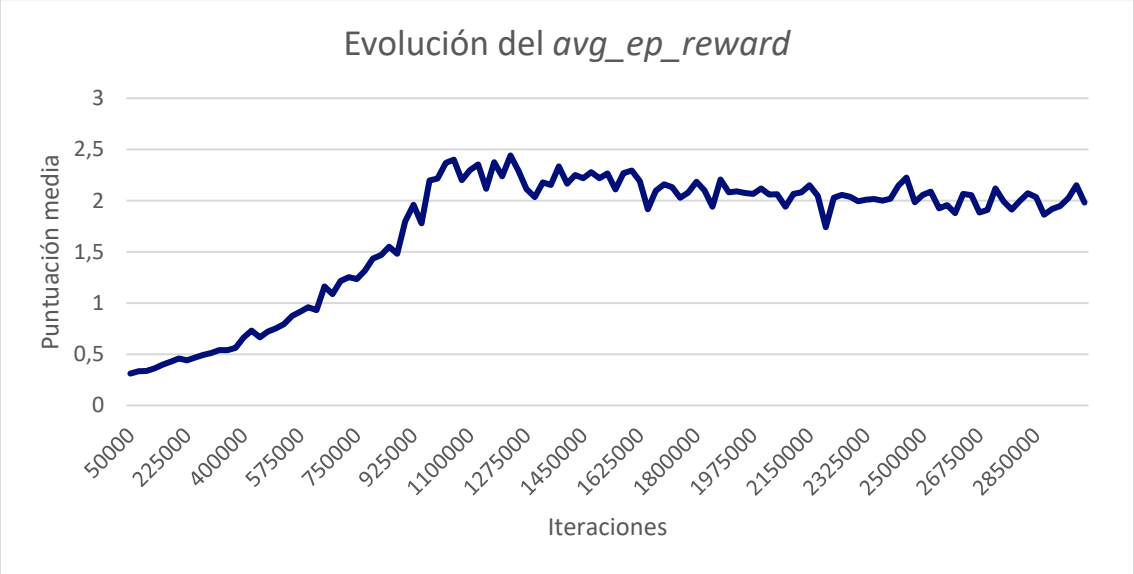


Ilustración 40. Evolución del avg_ep_reward para Breakout (etapa 2)

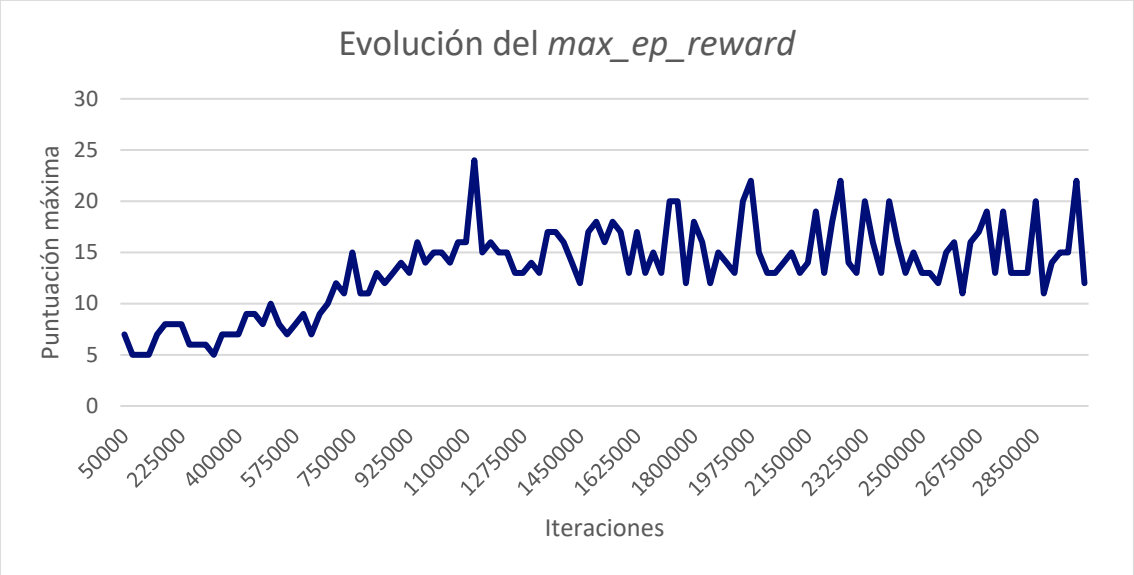


Ilustración 41. Evolución del max_ep_reward para Breakout (etapa 2)

7.3.2.2 *Space Invaders*

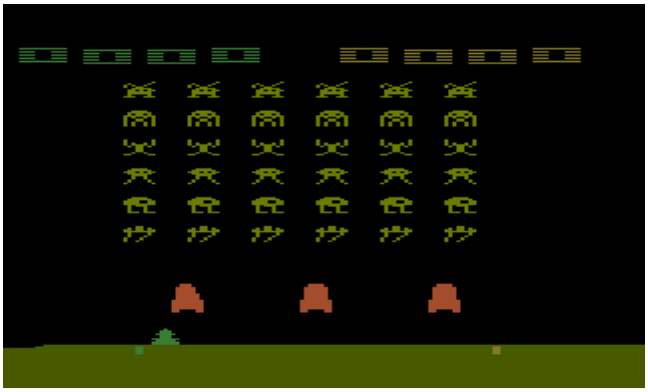


Ilustración 42. Captura de Space Invaders (obtenida de Iakupo [49])

Space Invaders (véase la Ilustración 42) tiene una mecánica muy parecida a *Breakout*, y al igual que él, posee un conjunto de 6 acciones. El objetivo es acabar con todos los marcianos sin que el jugador sea destruido por ellos. Se ha escogido este juego porque cumple con dos criterios deseados: (1) IW es capaz de resolverlo mejor que una red entrenada con DQN (2877 puntos frente a 1976±893 puntos), y (2) IW es superado por otras técnicas de *planning* expuestas en el artículo de referencia [17].

La duración del entrenamiento ha sido de **16 horas, 51 minutos y 26 segundos**. A continuación, se exponen las gráficas de evolución:

Ilustración 43. Evolución del avg_reward para Space Invaders (etapa 2)

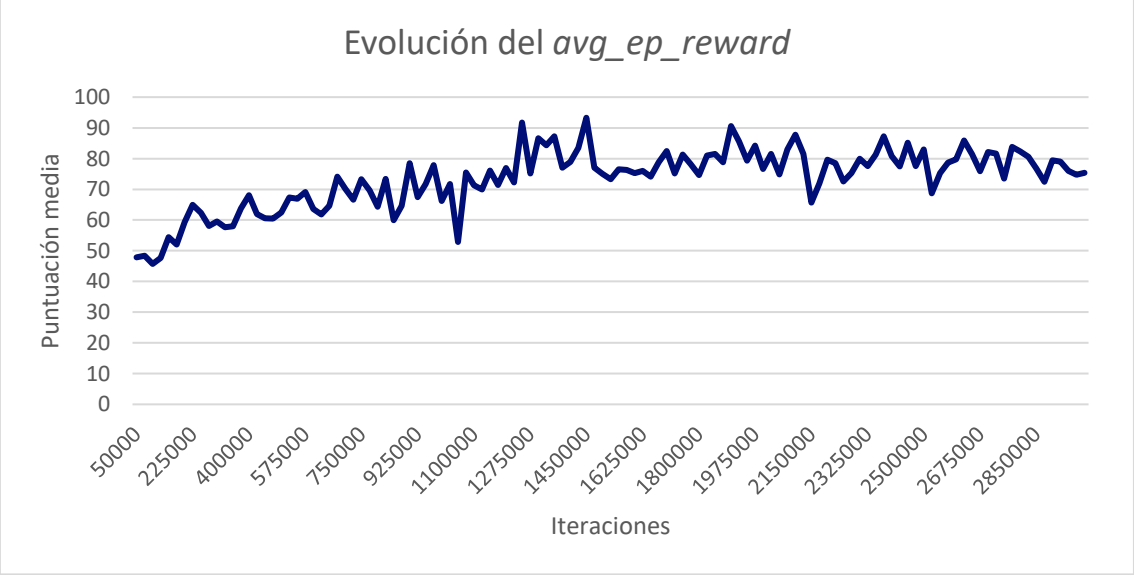


Ilustración 44. Evolución del avg_ep_reward para Space Invaders (etapa 2)

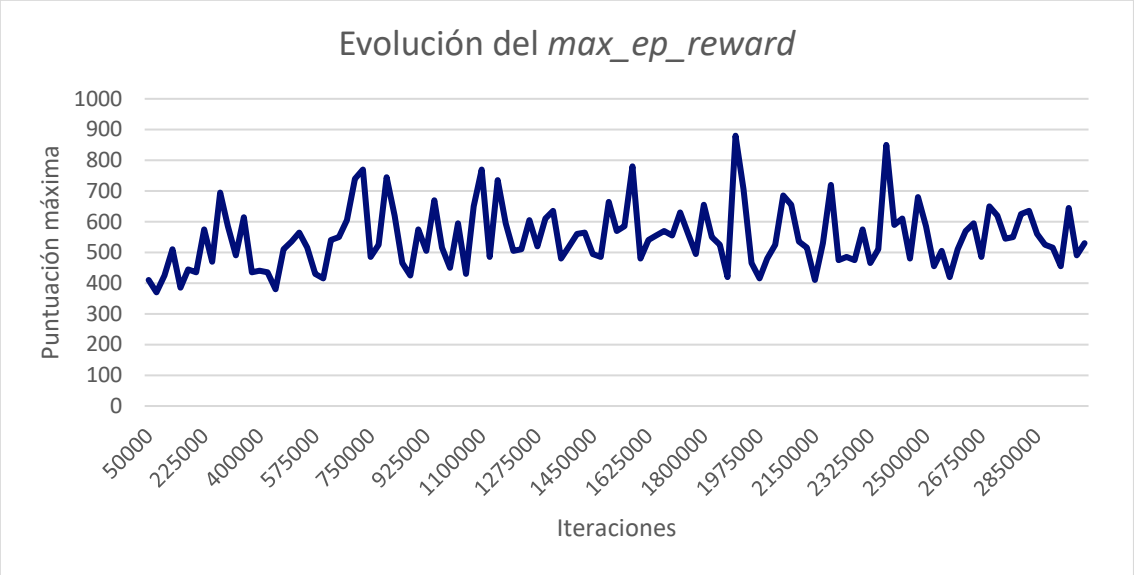


Ilustración 45. Evolución del max_ep_reward para Space Invaders (etapa 2)

7.3.2.3 *Beam Rider*

Ilustración 46. Captura de Beam Rider (obtenida de 8-Bit Central [50])

La mecánica de *Beam Rider* (véase la Ilustración 46) es ligeramente más compleja que la de los dos juegos anteriores (tiene un *set* de 9 acciones posibles). El objetivo de *Beam Rider* es limpiar 99 sectores de naves alienígenas. Para completar cada sector es necesario eliminar 15 naves alienígenas. Las decisiones que puede tomar el jugador es esquivar las naves, destruirlas con un láser ilimitado o destruirlas con un torpedo (limitados a 3 en cada sector).

Se ha escogido este juego porque cumple con dos criterios deseados: (1) IW es capaz de resolverlo mejor que una red entrenada con DQN (9108 puntos frente a 6846±1619 puntos), y (2) IW es superado por otras técnicas de *planning* expuestas en el artículo de referencia [17].

La duración del entrenamiento ha sido de **16 horas, 50 minutos y 29 segundos**. A continuación, se exponen las gráficas de evolución:

Ilustración 47. Evolución del avg_reward para Beam Rider (etapa 2)

Ilustración 48. Evolución del avg_ep_reward para Beam Rider (etapa 2)

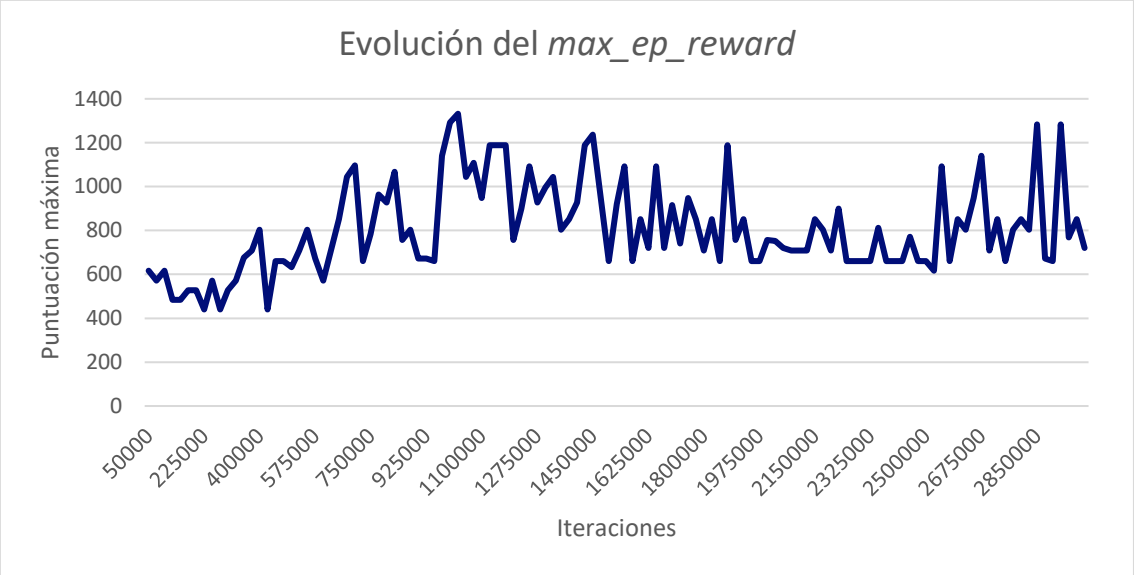


Ilustración 49. Evolución del max_ep_reward para Beam Rider (etapa 2)

7.3.2.4 *Frostbite*

Ilustración 50. Captura de Frostbite (obtenida de Gaga Games [51])

La dinámica de *Frostbite* (véase la Ilustración 50) es relativamente compleja hasta el punto de disponer de 18 acciones, que es el límite establecido para un juego de *Atari 2600*. El juego está compuesto por niveles. En cada nivel, el objetivo es construir un iglú con 15 bloques de hielo. Los bloques se obtienen saltando entre filas de placas de hielo que flotan sobre una masa de agua. El jugador debe evitar caer al agua y esquivar los enemigos. El jugador cuenta con 45 segundos para reunir los bloques y construir el iglú.

Se ha escogido este juego porque cumple con dos criterios deseados: (1) IW es capaz de resolverlo mejor que una red entrenada con DQN (902 puntos frente a 328.3±250.5 puntos), y (2) IW es superado por otras técnicas de *planning* expuestas en el artículo de referencia [17].

La duración del entrenamiento ha sido de **17 horas, 21 minutos y 16 segundos**. A continuación, se exponen las gráficas de evolución:

Ilustración 51. Evolución del avg_reward para Frostbite (etapa 2)

Ilustración 52. Evolución del avg_ep_reward para Frostbite (etapa 2)

Ilustración 53. Evolución del max_ep_reward para Frostbite (etapa 2)

7.3.2.5 *Q* Bert*

Ilustración 54. Captura de Q Bert (obtenida de Gizmodo [52])*

Q Bert* (véase la Ilustración 54) es un juego de dinámica sencilla que cuenta con un conjunto de 6 acciones posibles. El juego se desarrolla en una pirámide de cubos y está organizada en niveles. En cada nivel, el objetivo del jugador es conseguir que las superficies de todos los cubos cambien de color. Para ello, es necesario llegar hasta los cubos por medio de saltos en diagonal, esquivando a los enemigos y evitando caer de la pirámide.

Se ha escogido este juego porque cumple criterio deseado: IW lo resuelve peor que una red entrenada con DQN (3705 puntos frente a 10596±3294 puntos),

La duración del entrenamiento ha sido de **16 horas, 59 minutos y 16 segundos**. A continuación, se exponen las gráficas de evolución:

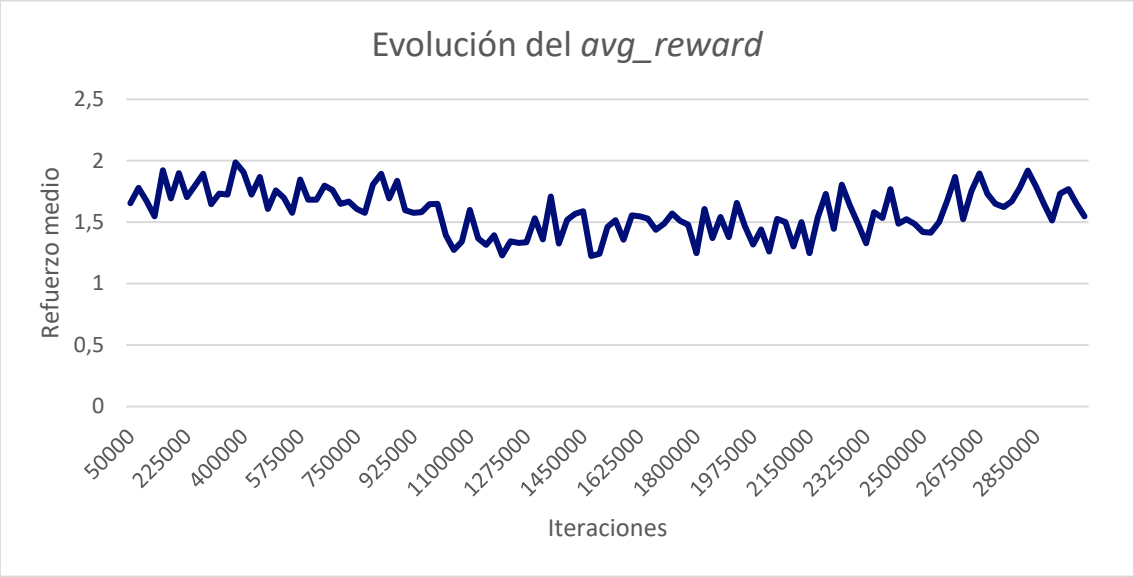


Ilustración 55. Evolución del avg_reward para Q Bert (etapa 2)*

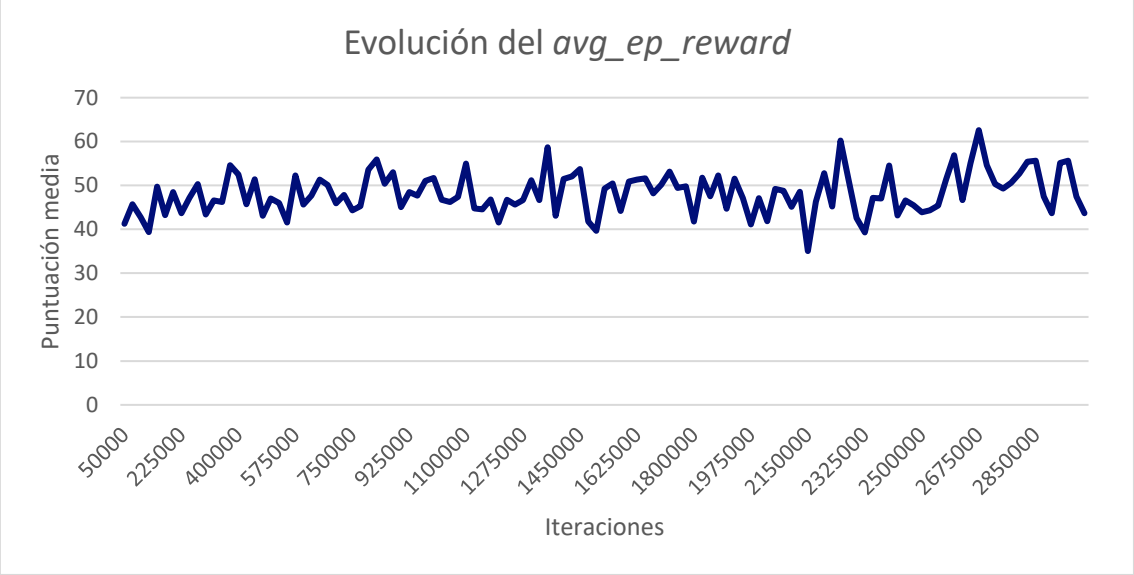


Ilustración 56. Evolución del avg_ep_reward para Q Bert (etapa 2)*

Ilustración 57. Evolución del max_ep_reward para Q Bert (etapa 2)*

7.3.2.6 *Star Gunner*

Ilustración 58. Captura de Star Gunner (obtenida de Atarimania [41])

Star Gunner (véase la Ilustración 58) cuenta con un conjunto completo de 18 acciones (al igual que *Frostbite*). Sin embargo, la mecánica del juego es muy simple. *Star Gunner* está compuesto por oleadas. En cada oleada, el jugador debe controlar una nave en un espacio bidimensional y disparar a los enemigos, esquivando sus disparos y evitando chocar con ellos.

Se ha escogido este juego porque cumple con dos criterios deseados: (1) IW lo resuelve peor que una red entrenada con DQN (1450 puntos frente a 57997±3152 puntos), y (2) IW es superado por otras técnicas de *planning* expuestas en el artículo de referencia [17].

La duración del entrenamiento ha sido de **15 horas, 48 minutos y 9 segundos**. A continuación, se exponen las gráficas de evolución:

Ilustración 59. Evolución del avg_reward para Star Gunner (etapa 2)

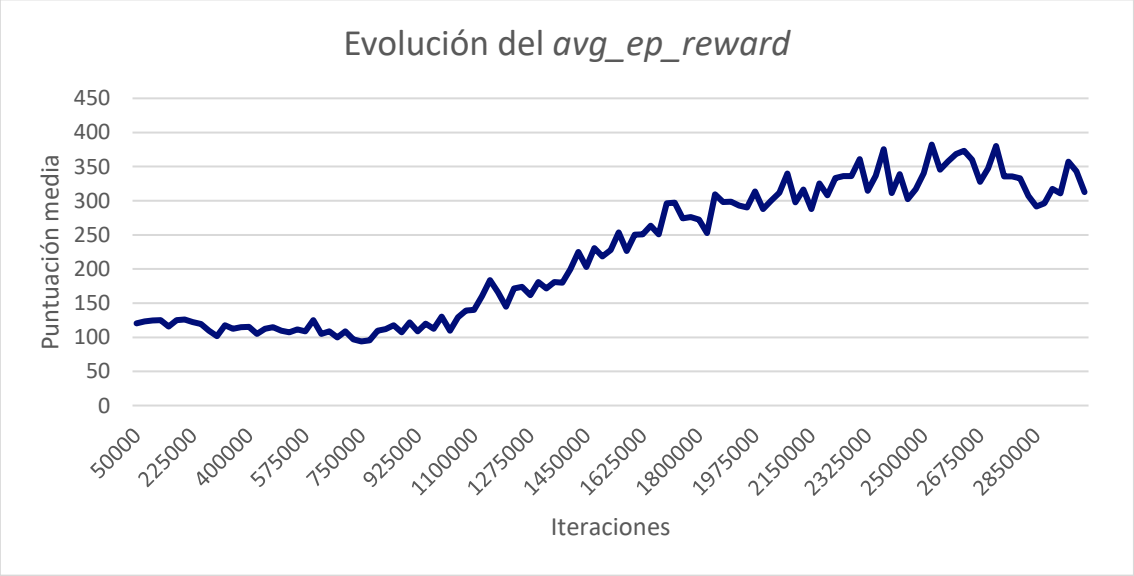


Ilustración 60. Evolución del avg_ep_reward para Star Gunner (etapa 2)

Ilustración 61. Evolución del max_ep_reward para Star Gunner (etapa 2)

7.3.3 Etapa 3: Experimentación con IW y las redes de DQN

Con la etapa 2 completada, se realizará la experimentación con los agentes. Las consideraciones globales de esta tercera etapa son las siguientes:

1. Se realizarán las mismas pruebas para cada uno de los juegos propuestos en la etapa 2. Cada prueba se repetirá 3 veces y se hará una media para evitar resultados poco confiables, dada la naturaleza no determinista de algunos experimentos.

2. Se ha establecido un límite de 8000 *frames* por cada partida, aunque en el caso de estudio sobre IW [17] era de 18000 *frames*. La razón es que algunos juegos (*Beam Rider*, por ejemplo) pueden alargarse demasiado en el tiempo. Como la cantidad de experimentos previstos en esta etapa es grande, se ha decidido limitar más aún el número de *frames* de cada experimento para reducir el tiempo total de experimentación. 8000 es un límite lo suficientemente elevado como para que los resultados sean admisibles.

3. Con cada experimento se ha extraído información relevante sobre la partida: puntuación, número de *frames* consumidos, tiempo de ejecución de la partida, tiempo medio por *frame*.

7.3.3.1 Fase 1: *SearchAgent* con algoritmo de planificación IW

En la Tabla 73 se muestran los valores de los parámetros involucrados en esta primera fase:

juego	*limite_simulacion*
• *Breakout* • *Space Invaders* • *Beam Rider* • *Frostbite* • *Q* Bert* • *Star Gunner*	• 1000 • 2500 • 5000 • 10000 • 25000 • 50000 • 75000 • 150000

Tabla 73. Parámetros y valores usados para la etapa 3, fase 1

Se ejecutará el agente *SearchAgent* con el algoritmo de planificación IW. El número de experimentos será igual a todas las posibles combinaciones de valores de la Tabla 73.

En el caso de estudio sobre IW [17] también realizaron una simulación con el agente *SearchAgent* y el algoritmo IW, pero únicamente con *limite_simulacion* igual a 150000, un valor muy alto que ralentiza demasiado los experimentos. En esta experimentación se probarán más valores de *limite_simulacion* por que los propios autores del artículo indican que valores altos no implican siempre puntuaciones altas, así que interesa averiguar cómo influye *limite_simulacion* en las puntuaciones.

Todos los experimentos de esta fase suman **59 horas, 26 minutos y 1 segundo** de ejecución. En los siguientes subapartados se incluyen las tablas con todos los experimentos:

7.3.3.1.1 Breakout

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	**0**	497	14,835	0,030
2500	**429**	7567	507,516	0,067
5000	**431**	6910	520,623	0,075
10000	**396**	7656	622,148	0,081
25000	**408**	6489	503,599	0,078
50000	**430**	7123	638,490	0,090
75000	**406**	8000	827,451	0,103
150000	**425**	7139	742,874	0,104

Tabla 74. Experimentación con Breakout y SearchAgent (etapa 3, fase 1)

7.3.3.1.2 Space Invaders

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	**340**	3883	117,324	0,030
2500	**495**	3221	216,650	0,067
5000	**465**	2701	339,459	0,126
10000	**2795**	7171	1732,820	0,242
25000	**2490**	7713	3637,737	0,472
50000	**2860**	8000	4005,590	0,501
75000	**2750**	7875	4072,353	0,517
150000	**1750**	5584	3675,803	0,658

Tabla 75. Experimentación con Space Invaders y SearchAgent (etapa 3, fase 1)

7.3.3.1.3 Beam Rider

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	**616**	5240	157,148	0,030
2500	**756**	4344	284,943	0,066
5000	**1380**	5406	689,743	0,128
10000	**2056**	6526	1651,447	0,253
25000	**2692**	8000	5010,030	0,626
50000	**2160**	8000	6248,457	0,781
75000	**2160**	8000	6345,883	0,793
150000	**2160**	8000	6958,953	0,870

Tabla 76. Experimentación con Beam Rider y SearchAgent (etapa 3, fase 1)

7.3.3.1.4 Frostbite

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	**100**	1743	57,439	0,033
2500	**190**	2030	145,660	0,072
5000	**250**	2456	343,119	0,140
10000	**300**	2330	629,369	0,270
25000	**4290**	4640	3014,933	0,650
50000	**230**	1940	1476,317	0,761

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
75000	**4330**	4600	4285,330	0,932
150000	**240**	2000	1632,027	0,816

Tabla 77. Experimentación con Frostbite y SearchAgent (etapa 3, fase 1)

7.3.3.1.5 Q Bert*

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	**775**	1771	47,514	0,027
2500	**775**	1817	112,047	0,062
5000	**1350**	2544	304,388	0,120
10000	**6850**	5096	1015,107	0,199
25000	**850**	2014	604,506	0,300
50000	**1400**	2985	937,640	0,314
75000	**5050**	3819	1074,603	0,281
150000	**1325**	2399	735,148	0,306

Tabla 78. Experimentación con Q Bert y SearchAgent (etapa 3, fase 1)*

7.3.3.1.6 Star Gunner

limite_simulacion	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	**900**	1939	38,024	0,020
2500	**400**	2055	88,357	0,043
5000	**1200**	2204	187,298	0,085
10000	**1000**	2027	343,829	0,170
25000	**1300**	1817	758,170	0,417
50000	**1500**	1048	825,210	0,787
75000	**1000**	991	1175,840	1,187
150000	**1000**	1002	1966,667	1,963

Tabla 79. Experimentación con Star Gunner y SearchAgent (etapa 3, fase 1)

7.3.3.2 Fase 2: *CombinationAgent* con algoritmo de planificación IW

En la Tabla 80 se muestran los valores de los parámetros involucrados en esta segunda fase:

juego	*modo*	*ratio*	*limite_simulacion*
• *Breakout* • *Space Invaders* • *Beam Rider* • *Frostbite* • *Q* Bert* • *Star Gunner*	• *Consensus* • *Probability*	• 0,25 • 0,375 • 0,5 • 0,625 • 0,75	• 1000 • 5000 • 25000

Tabla 80. Parámetros y valores usados para la etapa 3, fase 2

Se ejecutará el agente *CombinationAgent*, que combina el algoritmo de planificación IW con las redes de DQN. El número de experimentos será igual a todas las posibles combinaciones de valores de la Tabla 80.

En esta parte de la experimentación se tendrán en cuenta las siguientes consideraciones:

- Solo se experimentará con valores relativamente pequeños de *limite_simulacion* (1000, 5000 y 25000). La razón es que la experimentación con *SearchAgent* ha demostrado que no se necesitan valores altos para obtener buenas puntuaciones. Por ello, se han preferido valores pequeños para reducir todo lo posible el tiempo de experimentación, sin repercutir en la calidad de la misma.
- Un problema común en las redes convolucionales entrenadas con DQN en el ámbito de los juegos de Atari es que pueden darse bucles infinitos. Algunos juegos necesitan una acción inicial específica (denominada acción “FIRE”) que ponga en marcha la partida o un nivel dentro de la partida. Si la red, para ese estado inicial de la partida, no considera esa acción, la partida entra en bucle. Para evitar esta situación (que es común cuando *ratio* es mayor a 0.5 porque la red de DQN tiene más peso), se ha decidido que, por cada 5 acciones consensuadas conjuntamente por IW y la red de DQN, 1 acción sea elegida únicamente por IW.

Todos los experimentos con *modo Consensus* suman **66 horas, 14 minutos y 59 segundos** de ejecución, mientras que los experimentos con *modo Probability* suman **54 horas, 27 minutos y 9 segundos**. En los siguientes subapartados se incluyen las tablas con todos los experimentos:

7.3.3.2.1 Breakout

- ***modo Consensus***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**0**	497	17,202	0,035
	0,375	**0**	497	17,643	0,035
	0,5	**0**	497	17,895	0,036
	0,625	**0**	497	17,678	0,036
	0,75	**0**	497	18,062	0,036
5000	0,25	**429**	6100	535,447	0,088
	0,375	**419**	7891	794,445	0,101
	0,5	**323**	4931	444,244	0,090
	0,625	**12**	1897	240,986	0,127
	0,75	**5**	1422	168,890	0,119
25000	0,25	**376**	7937	797,573	0,100
	0,375	**362**	8000	865,956	0,108
	0,5	**6**	1598	278,395	0,174
	0,625	**8**	1949	362,293	0,186
	0,75	**10**	1999	367,675	0,184

Tabla 81. Experimentación con Breakout y CombinationAgent "Consensus" (etapa 3, fase 2)

- ***modo Probability***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**2,0**	840,3	28,143	0,033
	0,375	**1,3**	712,7	24,940	0,035
	0,5	**2,7**	934,7	33,580	0,036
	0,625	**1,0**	683,7	24,212	0,035
	0,75	**1,3**	767,0	28,243	0,037
5000	0,25	**32,3**	2560,7	335,352	0,131
	0,375	**24,7**	2922,7	349,894	0,120
	0,5	**10,0**	1874,7	259,187	0,138
	0,625	**10,0**	1743,3	227,239	0,130
	0,75	**7,3**	1651,7	197,285	0,119
25000	0,25	**5,3**	1441,0	240,075	0,167

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
	0,375	**8,3**	1801,7	315,695	0,175
	0,5	**8,7**	1897,0	314,059	0,166
	0,625	**5,7**	1347,7	222,419	0,165
	0,75	**8,3**	1620,0	297,056	0,183

Tabla 82. Experimentación con Breakout y CombinationAgent "Probability" (etapa 3, fase 2)

7.3.3.2.2 Space Invaders

- ***modo Consensus***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**165**	2569	94,012	0,037
	0,375	**165**	2569	93,846	0,037
	0,5	**165**	2569	92,885	0,036
	0,625	**50**	1623	58,057	0,036
	0,75	**210**	2563	93,270	0,036
5000	0,25	**325**	2373	326,427	0,138
	0,375	**830**	4917	667,817	0,136
	0,5	**155**	1725	237,774	0,138
	0,625	**445**	3431	475,269	0,139
	0,75	**105**	1575	215,628	0,137
25000	0,25	**2430**	8000	4281,580	0,535
	0,375	**2275**	8000	4171,220	0,521
	0,5	**1910**	7237	3809,700	0,526
	0,625	**315**	2547	1554,960	0,611
	0,75	**345**	2447	1603,450	0,655

Tabla 83. Experimentación con Space Invaders y CombinationAgent "Consensus" (etapa 3, fase 2)

- ***modo Probability***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**235,0**	2431,0	79,917	0,033
	0,375	**186,7**	2555,7	81,927	0,032
	0,5	**108,3**	2106,3	71,783	0,034
	0,625	**101,7**	1906,3	66,516	0,035

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
	0,75	**191,7**	2259,0	76,592	0,034
5000	0,25	**510,0**	3145,0	407,928	0,130
	0,375	**293,3**	2638,3	349,238	0,132
	0,5	**365,0**	3082,3	399,721	0,130
	0,625	**163,3**	2088,3	280,821	0,134
	0,75	**393,3**	3171,7	435,378	0,137
25000	0,25	**1535,0**	6685,0	3795,904	0,568
	0,375	**605,0**	3217,7	1816,530	0,565
	0,5	**456,7**	3869,0	2215,456	0,573
	0,625	**308,3**	3487,0	2056,959	0,590
	0,75	**228,3**	2418,3	1446,869	0,598

Tabla 84. Experimentación con Space Invaders y CombinationAgent "Probability" (etapa 3, fase 2)

7.3.3.2.3 Beam Rider

- ***modo Consensus***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**660**	5716	209,195	0,037
	0,375	**660**	5716	207,616	0,036
	0,5	**948**	8000	288,915	0,036
	0,625	**708**	8000	294,732	0,037
	0,75	**708**	8000	293,704	0,037
5000	0,25	**1380**	6110	861,507	0,141
	0,375	**1380**	6110	879,107	0,144
	0,5	**2160**	8000	1173,530	0,147
	0,625	**660**	3620	496,010	0,137
	0,75	**1536**	8000	1169,080	0,146
25000	0,25	**1380**	5208	3231,990	0,621
	0,375	**2160**	7052	4567,590	0,648
	0,5	**2160**	8000	5326,780	0,666
	0,625	**2160**	8000	5312,430	0,664
	0,75	**1092**	6608	4180,650	0,633

Tabla 85. Experimentación con Beam Rider y CombinationAgent "Consensus" (etapa 3, fase 2)

- ***modo Probability***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**486,7**	4846,0	149,438	0,031
	0,375	**868,0**	6998,7	219,660	0,031
	0,5	**868,0**	6340,7	202,584	0,032
	0,625	**736,0**	5667,3	181,927	0,032
	0,75	**724,0**	6480,0	214,749	0,033
5000	0,25	**1624,0**	6106,7	844,851	0,138
	0,375	**1400,0**	5813,3	778,190	0,134
	0,5	**1805,3**	6907,3	956,964	0,139
	0,625	**1624,0**	6832,7	941,092	0,138
	0,75	**1220,0**	5708,7	774,261	0,136
25000	0,25	**1865,3**	7272,7	4611,134	0,634
	0,375	**1441,3**	6274,0	3891,753	0,620
	0,5	**1934,7**	8000,0	5054,287	0,632
	0,625	**1744,0**	7876,7	4961,431	0,630
	0,75	**1348,0**	7628,0	4860,951	0,637

Tabla 86. Experimentación con Beam Rider y CombinationAgent "Probability" (etapa 3, fase 2)

7.3.3.2.4 Frostbite

- ***modo Consensus***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**250**	2621	105,113	0,040
	0,375	**250**	2621	105,135	0,040
	0,5	**250**	2621	104,118	0,040
	0,625	**230**	2190	87,354	0,040
	0,75	**100**	1626	64,106	0,039
5000	0,25	**250**	2030	317,385	0,156
	0,375	**260**	2340	364,142	0,156
	0,5	**270**	2380	360,412	0,151
	0,625	**270**	2306	339,095	0,147
	0,75	**170**	1806	274,417	0,152
25000	0,25	**4630**	5546	3608,930	0,651

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
	0,375	**4050**	5180	3560,410	0,687
	0,5	**290**	2441	1494,450	0,612
	0,625	**220**	2156	1323,740	0,614
	0,75	**240**	2360	1368,660	0,580

Tabla 87. Experimentación con Frostbite y CombinationAgent "Consensus" (etapa 3, fase 2)

- ***modo Probability***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**120,0**	1705,3	58,805	0,034
	0,375	**136,7**	1880,0	64,284	0,034
	0,5	**130,0**	1820,7	63,836	0,035
	0,625	**193,3**	2100,7	76,087	0,036
	0,75	**156,7**	1807,3	66,177	0,037
5000	0,25	**263,3**	2352,0	348,687	0,148
	0,375	**220,0**	2048,7	305,175	0,149
	0,5	**236,7**	2056,7	304,291	0,148
	0,625	**233,3**	2092,0	311,730	0,149
	0,75	**210,0**	2078,7	311,499	0,150
25000	0,25	**230,0**	2225,3	1358,021	0,610
	0,375	**190,0**	1936,7	1188,033	0,613
	0,5	**230,0**	2305,3	1398,623	0,607
	0,625	**186,7**	2028,0	1201,082	0,592
	0,75	**200,0**	2368,7	1429,063	0,603

Tabla 88. Experimentación con Frostbite y CombinationAgent "Probability" (etapa 3, fase 2)

7.3.3.2.5 Q Bert*

- ***modo Consensus***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**625**	1472	47,795	0,032
	0,375	**625**	1472	47,943	0,033
	0,5	**675**	1901	62,864	0,033
	0,625	**475**	1388	45,283	0,033

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
	0,75	**475**	1388	44,999	0,032
5000	0,25	**9550**	5179	672,896	0,130
	0,375	**5350**	4088	523,180	0,128
	0,5	**5775**	4204	535,796	0,127
	0,625	**9125**	5746	733,359	0,128
	0,75	**6850**	5629	706,198	0,125
25000	0,25	**5350**	4213	1241,280	0,295
	0,375	**6025**	4979	1357,350	0,273
	0,5	**1300**	2406	623,412	0,259
	0,625	**1600**	5151	1282,380	0,249
	0,75	**250**	1726	563,790	0,327

Tabla 89. Experimentación con Q Bert y CombinationAgent "Consensus" (etapa 3, fase 2)*

- ***modo Probability***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**550,0**	1508,7	41,926	0,028
	0,375	**575,0**	1664,7	45,623	0,027
	0,5	**616,7**	1776,7	51,183	0,029
	0,625	**566,7**	1714,0	49,789	0,029
	0,75	**525,0**	1682,7	49,642	0,030
5000	0,25	**3400,0**	3129,3	394,776	0,126
	0,375	**3641,7**	3036,0	381,870	0,126
	0,5	**5491,7**	4155,3	524,850	0,126
	0,625	**958,3**	2331,7	301,166	0,129
	0,75	**933,3**	2314,0	297,628	0,129
25000	0,25	**9466,7**	5675,7	1431,056	0,252
	0,375	**975,0**	2322,7	665,807	0,287
	0,5	**3891,7**	3544,3	1009,453	0,285
	0,625	**900,0**	2734,3	700,901	0,256
	0,75	**1033,3**	2668,7	831,941	0,312

Tabla 90. Experimentación con Q Bert y CombinationAgent "Probability" (etapa 3, fase 2)*

7.3.3.2.6 Star Gunner

- ***modo Consensus***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**2100**	3761	99,302	0,026
	0,375	**2100**	3761	99,501	0,026
	0,5	**2100**	3761	98,927	0,026
	0,625	**2800**	6007	161,074	0,027
	0,75	**2200**	4587	122,993	0,027
5000	0,25	**1500**	2987	277,314	0,093
	0,375	**1500**	2678	250,955	0,094
	0,5	**2000**	3161	295,542	0,093
	0,625	**1500**	2359	216,195	0,092
	0,75	**3200**	5785	544,745	0,094
25000	0,25	**1800**	1885	826,912	0,439
	0,375	**1300**	1694	715,317	0,422
	0,5	**1600**	2195	949,449	0,433
	0,625	**1500**	1787	782,571	0,438
	0,75	**4800**	4387	1942,320	0,443

Tabla 91. Experimentación con Star Gunner y CombinationAgent "Consensus" (etapa 3, fase 2)

- ***modo Probability***

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
1000	0,25	**833,3**	2341,3	47,931	0,020
	0,375	**1100,0**	2348,3	46,384	0,020
	0,5	**1166,7**	2902,0	61,276	0,021
	0,625	**1200,0**	2944,7	64,683	0,022
	0,75	**1600,0**	4123,3	93,580	0,023
5000	0,25	**1166,7**	1799,3	156,646	0,087
	0,375	**1266,7**	3529,0	321,982	0,091
	0,5	**1733,3**	2737,3	244,725	0,089
	0,625	**3033,3**	4010,7	366,741	0,091
	0,75	**1700,0**	3661,7	336,749	0,092
25000	0,25	**1400,0**	1582,3	650,379	0,411

limite_simulacion	*ratio*	Puntuación	*Frames*	Tiempo (s)	Tiempo por *frame* (s)
	0,375	**900,0**	1490,3	585,381	0,393
	0,5	**1400,0**	1686,0	692,292	0,411
	0,625	**2433,3**	2630,0	1108,709	0,422
	0,75	**2633,3**	3036,3	1260,090	0,415

Tabla 92. Experimentación con Star Gunner y CombinationAgent "Probability" (etapa 3, fase 2)

7.3.3.3 Resumen de puntuaciones

En los siguientes subapartados, se muestran tablas-resumen en la que se comparan las puntuaciones obtenidas en los experimentos de la etapa 3: fase 1 (*SearchAgent*), fase 2 (*CombinationAgent, modo Consensus* y *Probability*).

7.3.3.3.1 Breakout

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
1000	0	0	-	-
	0,25	-	0	2,0
	0,375	-	0	1,3
	0,5	-	0	2,7
	0,625	-	0	1,0
	0,75	-	0	1,3
5000	0	431	-	-
	0,25	-	429	32,3
	0,375	-	419	24,7
	0,5	-	323	10,0
	0,625	-	12	10,0
	0,75	-	5	7,3
25000	0	408	-	-
	0,25	-	376	5,3
	0,375	-	362	8,3
	0,5	-	6	8,7
	0,625	-	8	5,7
	0,75	-	10	8,3

Tabla 93. Resumen de la experimentación con Breakout

7.3.3.3.2 Space Invaders

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
1000	0	340	-	-
	0,25	-	165	235,0
	0,375	-	165	186,7
	0,5	-	165	108,3
	0,625	-	50	101,7
	0,75	-	210	191,7

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
5000	0	465	-	-
	0,25	-	325	510,0
	0,375	-	830	293,3
	0,5	-	155	365,0
	0,625	-	445	163,3
	0,75	-	105	393,3
25000	0	2490	-	-
	0,25	-	2430	1535,0
	0,375	-	2275	605,0
	0,5	-	1910	456,7
	0,625	-	315	308,3
	0,75	-	345	228,3

Tabla 94. Resumen de la experimentación con Space Invaders

7.3.3.3.3 Beam Rider

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
1000	0	616	-	-
	0,25	-	660	486,7
	0,375	-	660	868,0
	0,5	-	948	868,0
	0,625	-	708	736,0
	0,75	-	708	724,0
5000	0	1380	-	-
	0,25	-	1380	1624,0
	0,375	-	1380	1400,0
	0,5	-	2160	1805,3
	0,625	-	660	1624,0
	0,75	-	1536	1220,0
25000	0	2692	-	-

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
	0,25	-	1380	1865,3
	0,375	-	2160	1441,3
	0,5	-	2160	1934,7
	0,625	-	2160	1744,0
	0,75	-	1092	1348,0

Tabla 95. Resumen de la experimentación con Beam Rider

7.3.3.3.4 Frostbite

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
1000	0	100	-	-
	0,25	-	250	120,0
	0,375	-	250	136,7
	0,5	-	250	130,0
	0,625	-	230	193,3
	0,75	-	100	156,7
5000	0	250	-	-
	0,25	-	250	263,3
	0,375	-	260	220,0
	0,5	-	270	236,7
	0,625	-	270	233,3
	0,75	-	170	210,0
25000	0	4290	-	-
	0,25	-	4630	230,0
	0,375	-	4050	190,0
	0,5	-	290	230,0
	0,625	-	220	186,7
	0,75	-	240	200,0

Tabla 96. Resumen de la experimentación con Frostbite

7.3.3.3.5 Q Bert*

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
1000	0	775	-	-
	0,25	-	625	550,0
	0,375	-	625	575,0
	0,5	-	675	616,7
	0,625	-	475	566,7
	0,75	-	475	525,0
5000	0	1350	-	-
	0,25	-	9550	3400,0
	0,375	-	5350	3641,7
	0,5	-	5775	5491,7
	0,625	-	9125	958,3
	0,75	-	6850	933,3
25000	0	850	-	-
	0,25	-	5350	9466,7
	0,375	-	6025	975,0
	0,5	-	1300	3891,7
	0,625	-	1600	900,0
	0,75	-	250	1033,3

Tabla 97. Resumen de la experimentación con Q Bert*

7.3.3.3.6 Star Gunner

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
1000	0	900	-	-
	0,25	-	2100	833,3
	0,375	-	2100	1100,0
	0,5	-	2100	1166,7
	0,625	-	2800	1200,0
	0,75	-	2200	1600,0

limite_simulacion	*ratio*	Puntuación - Agentes		
		SearchAgent (IW)	*CombinationAgent* (*modo Consensus*)	*CombinationAgent* (*modo Probability*)
5000	0	1200	-	-
	0,25	-	1500	1166,7
	0,375	-	1500	1266,7
	0,5	-	2000	1733,3
	0,625	-	1500	3033,3
	0,75	-	3200	1700,0
25000	0	1300	-	-
	0,25	-	1800	1400,0
	0,375	-	1300	900,0
	0,5	-	1600	1400,0
	0,625	-	1500	2433,3
	0,75	-	4800	2633,3

Tabla 98. Resumen de la experimentación con Star Gunner

8 Análisis de los resultados

8.1 Etapa 1

Con los experimentos realizados en esta etapa puede asegurarse que la implementación de DQN es capaz de entrenar agentes para que se comporten inteligentemente. Las gráficas de evolución del *avg_reward* y *avg_ep_reward* demuestran que, al menos en *Breakout*, el entrenamiento permite al agente aprender una estrategia dentro de la partida para conseguir el mayor número de puntos.

En cuanto al número de experimentos realizados, son pocos por tres razones: (1) realizar cada experimento requeriría demasiados días de ejecución ininterrumpida y el proyecto cuenta con una limitaciones temporales y de *hardware* ya mencionadas en la etapa 1, (2) la implementación original de DQN no necesita muchas mejoras, y (3) la configuración de parámetros base de la Tabla 71 ha sido extraída del caso de estudio sobre DQN [16], que fue el resultado de una experimentación por parte de los autores.

Aunque la experimentación de esta etapa no haya sido del todo exhaustiva, los resultados son más que aceptables de cara a posteriores etapas.

8.2 Etapa 2

De los resultados de la etapa 2 puede comentarse lo siguiente:

- Las redes entrenadas con DQN resuelven los juegos con mayor puntuación que un agente aleatorio, como se observa en la Tabla 99 y la Tabla 100. Esto demuestra que DQN ha conseguido entrenar un agente inteligente para cada videojuego.
- Sin embargo, el entrenamiento con DQN en esta experimentación no ha sido tan productivo como el entrenamiento expuesto en el caso de estudio sobre DQN [16]. Como puede verse en la Tabla 99 y la Tabla 100, las puntuaciones obtenidas en el artículo son visiblemente más altas que las de esta experimentación. Aun así, las redes son útiles para la etapa 3 de la experimentación.
- Un problema que se observa en algunos videojuegos es el estancamiento del entrenamiento. Es especialmente evidente en *Breakout* y *Frostbite*, donde se alcanza demasiado pronto un máximo en el refuerzo medio por episodio que el entrenamiento no es capaz de superar. También se intuye un estancamiento temprano en *Beam Rider*, pero parece solucionarse en la fase final del entrenamiento. Por ello, es posible que solo se necesite más tiempo para que las redes convolucionales salgan del estancamiento y continúen aprendiendo.
- El peor entrenamiento ha sido el de *Q* Bert*. Las gráficas demuestran que la red apenas ha conseguido aprender. De hecho, en las gráficas *avg_reward* y

avg_ep_reward se observan unas ondulaciones que parecen indicar que la red olvida lo aprendido en iteraciones previas.

- El mejor entrenamiento ha sido el de *Star Gunner*. El aprendizaje parece ser continuo y sin estancamiento, y en las últimas iteraciones del entrenamiento la el *avg_ep_reward* casi cuadruplica el *avg_ep_reward* al inicio.

Videojuego	Puntuación - Agentes		
	Red de DQN (etapa 2)	Red de DQN (caso de estudio [16])	Aleatorio
Breakout	14,3	401,2	1,6
Space Invaders	292,4	1976,0	140,2
Beam Rider	1149,8	6846,0	326,2
Frostbite	153,1	328,8	61,7
Q Bert*	65,1	10596,0	151,3
Star Gunner	2146,5	57997,0	577,9

Tabla 99. Comparativa de puntuación media entre agentes de redes DQN y agente aleatorio

Videojuego	Puntuación - Agentes	
	Red de DQN (etapa 2)	Aleatorio
Breakout	34	7
Space Invaders	860	565
Beam Rider	2440	852
Frostbite	220	150
Q Bert*	550	600
Star Gunner	6000	1200

Tabla 100. Comparativa de puntuación máxima entre agente de redes DQN y agente aleatorio

8.3 Etapa 3

Como la etapa 3 se divide en dos fases, primero se comentarán los resultados de la fase 1:

- Contra todo pronóstico, *SearchAgent* con algoritmo de planificación IW funciona mejor de lo esperado con valores de *limite_simulacion* pequeños, lo que significa que IW puede resolver partidas con buenas puntuaciones sin llevar a cabo extensas y pesadas simulaciones del futuro, que necesitarían mucho tiempo de ejecución.

- La posible causa de que valores más altos de *limite_simulacion* no conlleven puntuaciones más altas es que los refuerzos de un futuro muy lejano no son relevantes para que el agente elija la mejor acción. Más bien interesan los refuerzos futuros a corto y medio plazo, que sirven para determinar qué impacto tendrían las acciones posibles si se ejecutasen en el estado actual.

Ahora se comentarán los resultados de la fase 2:

- Con el ***modo Consensus***, se han observado algunos comportamientos:
  - En videojuegos como *Breakout*, *Space Invaders* o *Frostbite* un *ratio* menor o igual a 0.5 proporciona las mejores puntuaciones. En estos casos, queda claro que las redes de DQN sirven de gran ayuda para que IW tome mejores decisiones, pero siempre que la opinión de las redes no tenga excesivo peso en la decisión final.
  - En *Star Gunner* sucede lo contrario, es decir, un *ratio* de 0.625 o 0.75 consigue las mejores puntuaciones. Esto se debe a que la red de DQN para este videojuego está muy bien entrenada, por lo que las puntuaciones son más altas cuando la red tiene mucho peso en la decisión conjunta con IW.
- Con el ***modo Probability*** las puntuaciones son más irregulares que con el *modo Consensus*. Posiblemente se deba al factor aleatorio que interviene en *Probability,* y que provoca variaciones de puntuación sustanciales para distintas ejecuciones de un mismo experimento. El factor aleatorio puede ser interesante si se desea que el agente se comporte de manera diferente en cada partida. Esto no sería posible con el *modo Consensus*.
- Se observa que las puntuaciones del *modo Probability* son generalmente más bajas que las de *Consensus*. El *modo Consensus* parece más efectivo que *Probability*, lo que significaría que es preferible que las acciones sean consensuadas entre IW y la red convolucional.
- El valor de *limite_simulacion* más adecuado es 5000 porque proporciona muy buenas puntuaciones en todos los videojuegos, y el tiempo medio por *frame* es muy reducido. Un valor 1000 consigue que el tiempo medio por *frame* sea muy bajo, pero a cambio de unas simulaciones de IW poco exhaustivas que se traducen en malas decisiones durante la partida. Un valor 25000 proporciona buenas puntuaciones, pero el tiempo medio por *frame* se dispara, lo que afectaría a la fluidez de la partida.

De la comparativa entre los resultados de la fase 1 y la fase 2 se puede decir que:

- La eficacia de la combinación de IW y las redes de DQN varía según el videojuego empleado.

- Para *Breakout*, *Space Invaders* y *Beam Rider*, *SearchAgent* se desenvuelve mejor que *CombinationAgent*. Sin embargo, las mejores puntuaciones de la combinación IW+DQN no están muy por debajo de las puntuaciones de IW por lo que el nuevo agente sigue siendo una buena alternativa (véase la Tabla 101).
- Para *Frostbite*, *Q* Bert* y *Star Gunner*, *CombinationAgent* es generalmente más efectivo que *SearchAgent*. Sobre todo, destacan *Q* Bert* y *Star Gunner*, donde *CombinationAgent* llega a multiplicar por 7 y por 4 las mejores puntuaciones de *SearchAgent*. Para estos 3 videojuegos, *CombinationAgent* también es mejor que las redes de DQN entrenadas en la etapa 2 de la experimentación (véase la Tabla 101).

Videojuego	**Puntuación - Agentes**		
	Redes de DQN (etapa 2)	*SearchAgent* (IW) (etapa 3)	*CombinationAgent* (etapa 3)
Breakout	14,3	431	429
Space Invaders	292,4	2490	2430
Beam Rider	1149,8	2692	2160
Frostbite	153,1	4290	4630
Q Bert*	65,1	1350	9550
Star Gunner	2146,5	1300	4800

Tabla 101. Puntuaciones más destacadas obtenidas con SearchAgent, redes de DQN y CombinationAgent

- El tiempo medio requerido para ejecutar un *frame* no varía entre *SearchAgent* y *CombinationAgent* porque el tiempo invertido por *frame* en utilizar las redes convolucionales es insignificante en comparación con el tiempo invertido por *frame* en la simulación de IW.

En la Tabla 102 se resumen las puntuaciones de los casos de estudio sobre DQN [16], IW [17] y CGP [18] para los 6 videojuegos propuestos:

Videojuego	*Planning*				*Reinforcement Learning*	*Genetic Programming*
	IW	2BFS	CGP	UCT	DQN	CGP
Breakout	384	772	1	364	401,2±26,9	13.2±2
Space Invaders	2877	3974	112	2718	1976±893	1001±25
Beam Rider	9108	9298	694	6625	6846±1619	1341.6±21
Frostbite	902	2672	137	271	328,3±250,5	782±795
Q Bert*	3705	11680	133	17343	10596±3294	770±94
Star Gunner	1540	4660	1345	1207	57997±3152	2320±303

Tabla 102. Puntuaciones de los casos de estudio sobre DQN [16], IW [17] y CGP [18]

Si lo comparamos con las puntuaciones de *CombinationAgent* de la Tabla 101, puede verse cómo el nuevo agente supera los resultados del resto de técnicas para *Breakout* y *Frostbite*. Para los demás videojuegos, *CombinationAgent* se mantiene entre las mejores opciones excepto en *Beam Rider*, donde las puntuaciones han sido visiblemente bajas, posiblemente por la limitación de 8000 *frames* por partida.

Aun así, comparar los resultados de esta experimentación con los resultados de los casos de estudio no es muy relevante puesto que los experimentos se han llevado a cabo en condiciones diferentes debido a las limitaciones de *hardware* y de tiempo mencionadas a lo largo de la experimentación. Los resultados de los casos de estudio de la Tabla 102 sirvieron únicamente como referencia para determinar qué videojuegos se podrían beneficiar con el nuevo agente *CombinationAgent*. Las comparaciones entre las fases 1 y 2 (véase la Tabla 101) sí son relevantes, porque los experimentos sí se realizaron en las mismas condiciones.

Las redes de DQN de *DeepMind* [16] no son comparables con las redes entrenadas en la etapa 2 de la experimentación porque los entrenamientos fueron muy diferentes. Lo mismo sucede con IW: los resultados del caso de estudio [17] son muy distintos a los resultados obtenidos con *SearchAgent* y *CombinationAgent* en la etapa 3 porque se escogieron valores de *limite_simulacion* más pequeños para reducir el tiempo de experimentación.

Curiosamente, esta reducción del valor de *limite_simulacion* consiguió que juegos como *Frostbite*, *Q* Bert* y *Star Gunner* obtuviesen puntuaciones más altas que las expuestas en el caso de estudio sobre IW [17]. Es posible que los refuerzos de un futuro muy lejano

no sean relevantes para que el agente elija la mejor acción, por lo que un valor alto de *limite_simulacion* puede no ser apropiado. Podrían ser preferibles los refuerzos de un futuro más inmediato, algo que se consigue con simulaciones de IW menos exhaustivas, es decir, valores bajos de *limite_simulacion*.

9 Planificación del proyecto

La duración estimada del proyecto son 6 meses, comenzando en el mes de febrero de 2018 y terminando en el mes de julio del mismo año. Se ha planificado para que se desarrolle en 5 fases:

- **Fase 1.** Fase inicial en la que se analiza en profundidad el tema del proyecto. Involucra tareas como el estudio de los fundamentes teóricos, herramientas *software* y casos de estudio de interés. Como resultado de la investigación, la fase concluye con la definición de los objetivos del proyecto.
- **Fase 2.** En esta fase se desarrollará el *software* necesario para cubrir los objetivos del trabajo. La fase consta de las etapas características de cualquier proyecto de desarrollo *software*: análisis, diseño, pruebas, mantenimiento, etc.
- **Fase 3.** El *software* construido en la fase anterior se somete a una experimentación. Se plantean tareas como la definición de la metodología de experimentación que se seguirá, descripción de parámetros de experimentación involucrados, ejecución de los experimentos y análisis de los resultados.
- **Fase 4.** La cuarta fase está orientada a la redacción de puntos que no podían completarse hasta terminar las fases anteriores (introducción, conclusiones, resumen) y puntos complementarios sobre cuestiones económicas, éticas y legales.
- **Fase continua.** Esta fase se desarrolla durante gran parte de la vida del proyecto y engloba tareas gestión del documento como bibliografía, glosario, formato; y tareas de mantenimiento del *software*.

A continuación, se muestra la planificación de las fases del proyecto en forma de tabla y de diagramas de Gantt:

Tarea	Inicio estimado	Fin estimado	Tiempo estimado	Inicio real	Fin real	Tiempo real
FASE 1	**08/02/2018**	**24/03/2018**	**45 días**	**08/02/2018**	**23/03/2018**	**44 días**
Organización de la fase	08/02/2018	09/02/2018	2 días	08/02/2018	09/02/2018	2 días
Estado del arte	10/02/2018	21/03/2018	40 días	10/02/2018	19/03/2018	38 días
Estudio de fundamentos teóricos	10/02/2018	16/02/2018	7 días	10/02/2018	16/02/2018	7 días
Casos de estudio	17/02/2018	02/03/2018	14 días	17/02/2018	05/03/2018	17 días
Estudio de herramientas *software*	03/03/2018	21/03/2018	19 días	06/03/2018	21/03/2018	16 días
Planteamiento de la cuestión del proyecto	22/03/2018	24/03/2018	3 días	22/03/2018	23/03/2018	2 días
FASE 2	**25/03/2018**	**11/05/2018**	**48 días**	**24/03/2018**	**31/05/2018**	**69 días**
Organización de la fase	25/03/2018	26/03/2018	2 días	25/03/2018	25/03/2018	1 día
Desarrollo *software*	27/03/2018	11/05/2018	46 días	26/03/2018	31/05/2018	67 días
Definición de la metodología	27/03/2018	30/03/2018	4 días	26/03/2018	29/03/2018	4 días
Análisis	31/03/2018	11/04/2018	12 días	30/03/2018	15/04/2018	17 días
Diseño	12/04/2018	23/04/2018	12 días	14/04/2018	29/04/2018	16 días
Implementación	24/04/2018	01/05/2018	8 días	30/04/2018	20/05/2018	21 días
Pruebas	02/05/2018	09/05/2018	8 días	21/05/2018	31/05/2018	11 días
Verificación	10/05/2018	11/05/2018	2 días	31/05/2018	31/05/2018	1 día
FASE 3	**12/05/2018**	**29/06/2018**	**49 días**	**01/06/2018**	**15/07/2018**	**45 días**
Organización de la fase	12/05/2018	13/05/2018	2 días	01/06/2018	02/06/2018	2 días
Evaluación de la cuestión	14/05/2018	29/06/2018	47 días	03/06/2018	15/07/2018	43 días

Tarea	Inicio estimado	Fin estimado	Tiempo estimado	Inicio real	Fin real	Tiempo real
Planteamiento de la metodología	14/05/2018	16/05/2018	3 días	03/06/2018	04/06/2018	2 días
Desarrollo de la experimentación	17/05/2018	24/06/2018	39 días	05/06/2018	15/07/2018	41 días
Análisis de los resultados	25/06/2018	29/06/2018	5 días	07/07/2018	15/07/2018	9 días
FASE 4	**30/06/2018**	**11/07/2018**	**12 días**	**30/06/2018**	**03/08/2018**	**35 días**
Organización de la fase	30/06/2018	01/07/2018	2 días	30/06/2018	30/06/2018	1 día
Marco regulador	02/07/2018	02/07/2018	1 día	01/07/2018	03/07/2018	3 días
Entorno socioeconómico	03/07/2018	04/07/2018	2 días	04/07/2018	06/07/2018	3 días
Introducción	05/07/2018	05/07/2018	1 día	07/07/2018	08/07/2018	2 días
Conclusión	06/07/2018	06/07/2018	1 día	09/07/2018	10/07/2018	2 días
Resumen del proyecto	07/07/2018	11/07/2018	5 días	27/07/2018	03/08/2018	8 días
FASE CONTINUA	**08/02/2018**	**11/07/2018**	**154 días**	**08/02/2018**	**03/08/2018**	**177 días**
Gestión de la memoria	08/02/2018	11/07/2018	154 días	08/02/2018	03/08/2018	177 días
Bibliografía	08/02/2018	11/07/2018	154 días	08/02/2018	03/08/2018	177 días
Glosario	08/02/2018	11/07/2018	154 días	08/02/2018	03/08/2018	177 días
Apéndices	08/02/2018	11/07/2018	154 días	08/02/2018	03/08/2018	177 días
Formato del documento	08/02/2018	11/07/2018	154 días	08/02/2018	03/08/2018	177 días
Gestión del *software*	12/05/2018	11/07/2018	61 días	01/06/2018	03/08/2018	64 días
Mantenimiento	12/05/2018	11/07/2018	61 días	01/06/2018	03/08/2018	64 días

Tabla 103. Planificación de las fases del proyecto

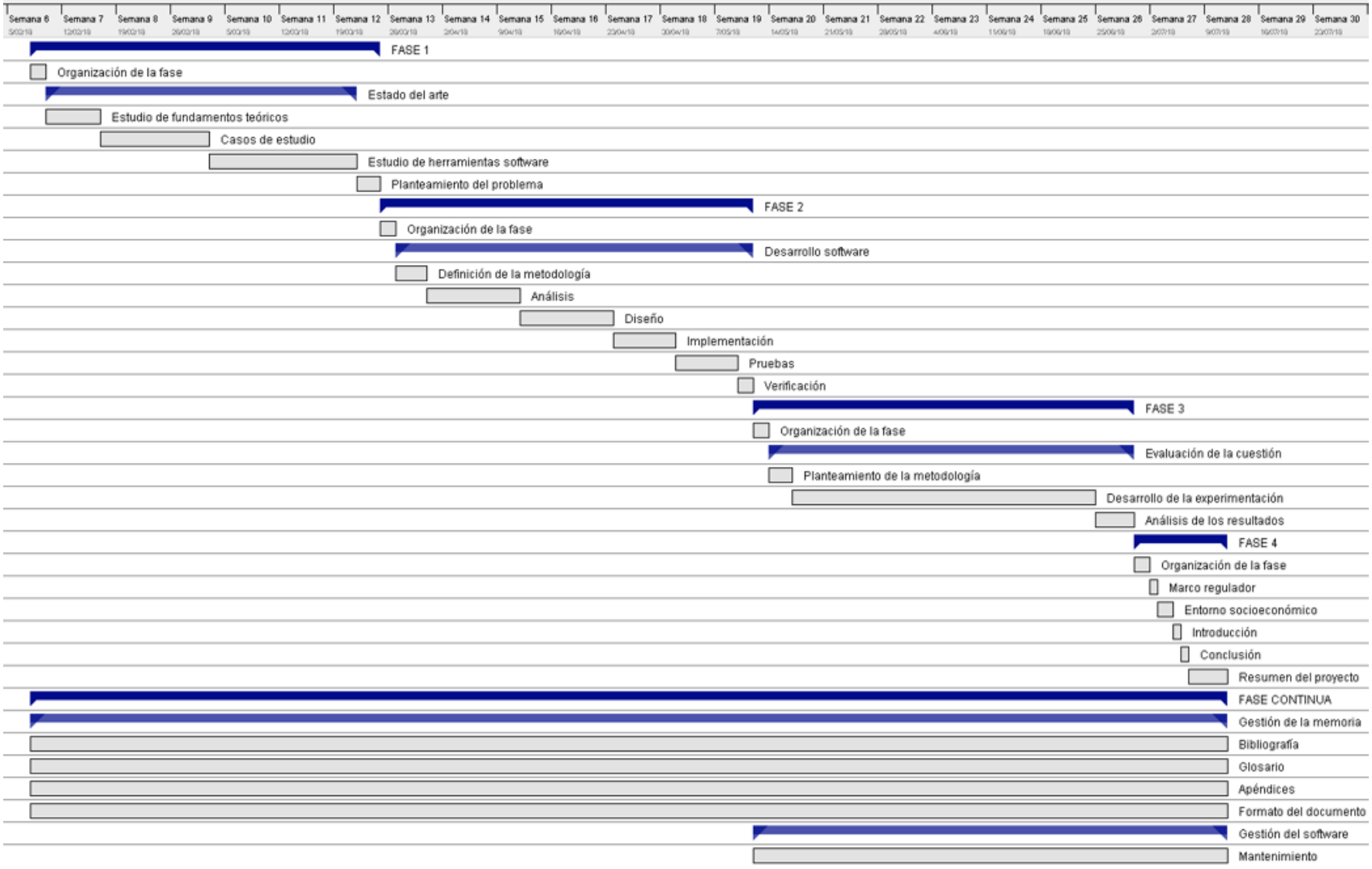


Ilustración 62. Diagrama de Gantt expandido

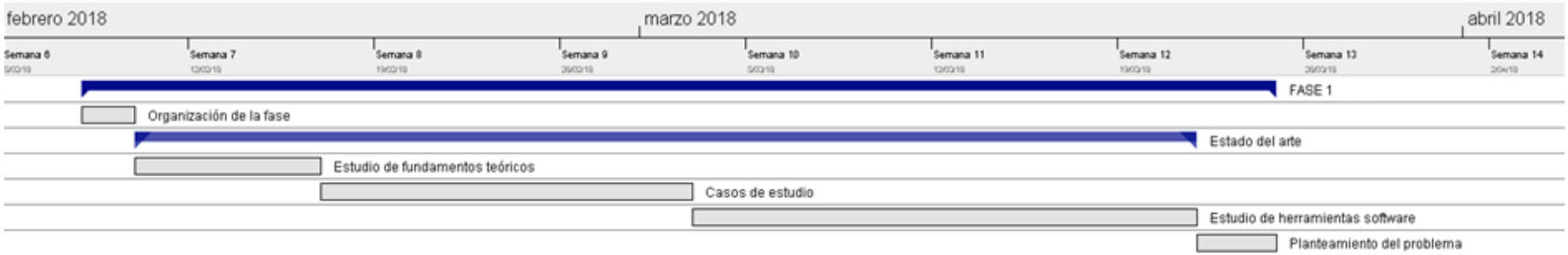


Ilustración 63. Diagrama de Gantt de fase 1

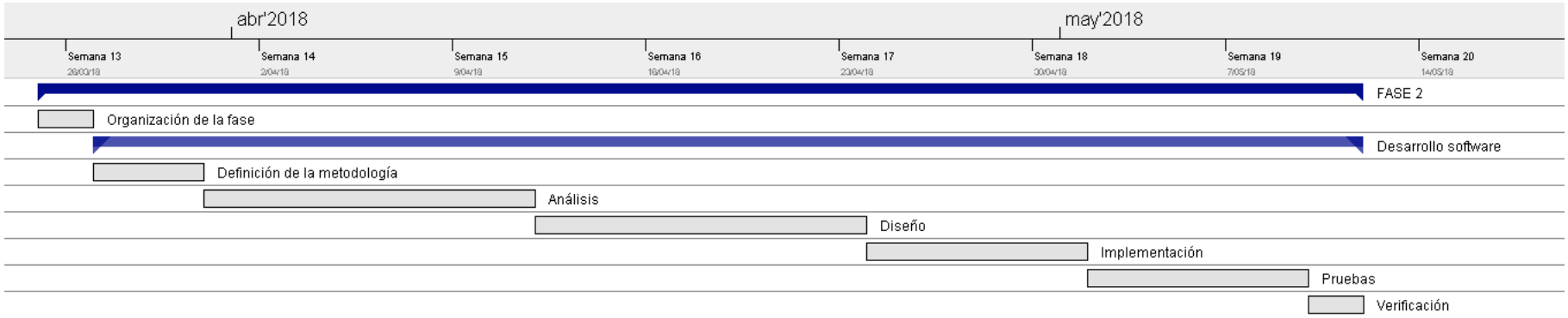


Ilustración 64. Diagrama de Gantt de fase 2

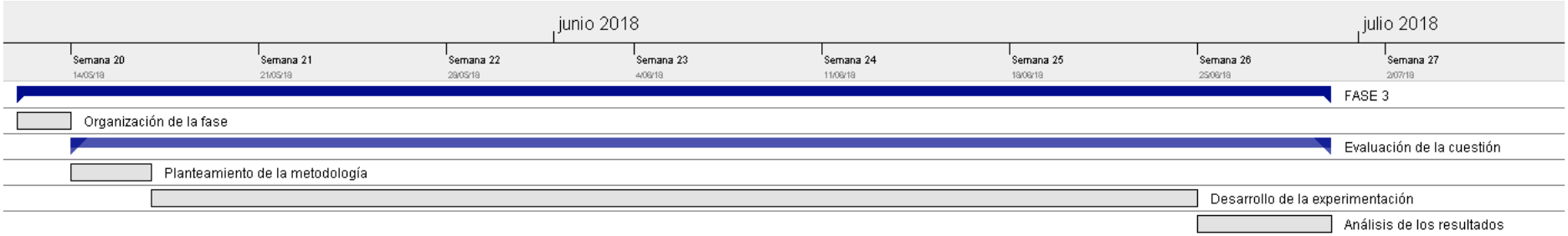


Ilustración 65. Diagrama de Gantt de fase 3

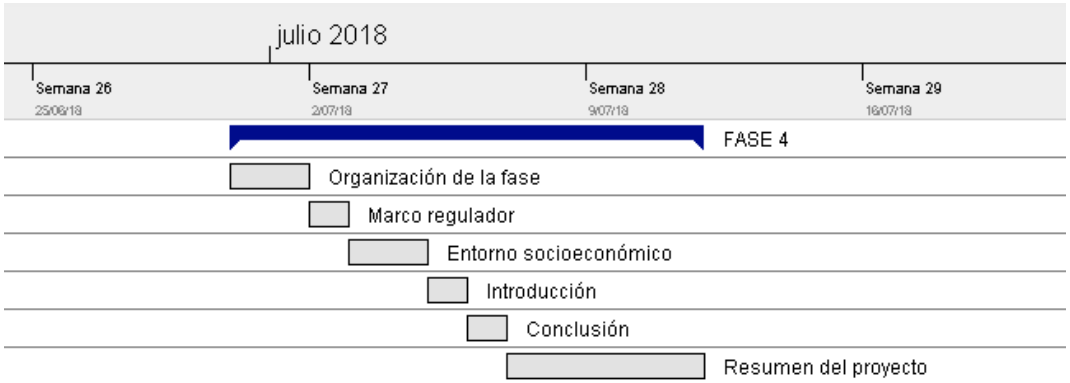


Ilustración 66. Diagrama de Gantt de fase 4

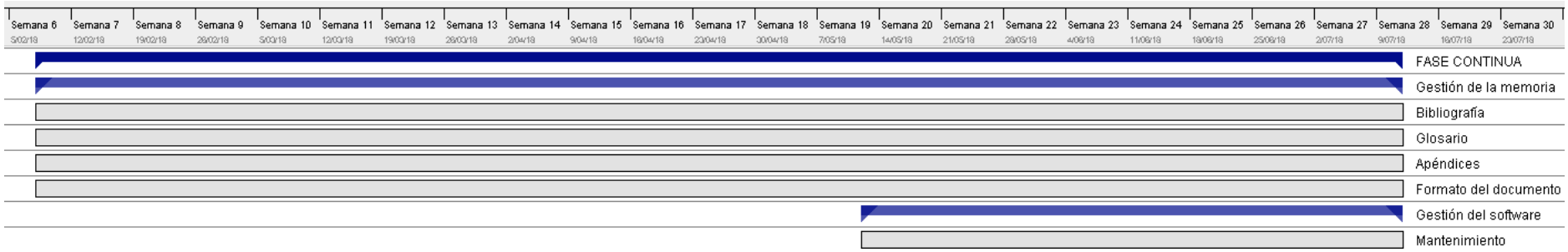


Ilustración 67. Diagrama de Gantt de fase continua

10 Entorno socioeconómico

10.1 Presupuesto del proyecto

En este apartado se detalla el presupuesto del proyecto, presentando en tablas los recursos utilizados y los costes imputables a cada recurso. En el caso de los recursos *hardware* y *software*, el coste imputable se ha obtenido con la siguiente fórmula:

$$coste\ imputable = \frac{coste\ del\ recurso * tiempo\ de\ uso}{periodo\ de\ amortización}$$

Los costes indirectos del presupuesto (electricidad, internet, etc.) representan el 15% del coste imputable la mano de obra.

- **Mano de obra**

Asumiendo que la persona trabajará a media jornada durante los 6 meses de proyecto, el número de horas dedicadas estimadas será de 480 horas.

Trabajador	Coste por hora	Tiempo dedicado	Coste imputable
Jorge Carmena Plaza	12 €/h	480 horas	5760 €
		Total	**5760 €**

Tabla 104. Detalle de mano de obra

- **Recursos *hardware***

Nombre	Unidades	Coste	Tiempo de uso	Periodo de amortización	Coste imputable
Portátil Asus N552VX	1	777,99 €	6 meses	48 meses	97,25 €
Portátil Samsung RV510	1	405 €	6 meses	48 meses	50,625 €
				Total	**147,875 €**

Tabla 105. Detalle de recursos hardware

- **Recursos *software***

Nombre	Unidades	Coste	Tiempo de uso	Periodo de amortización	Coste imputable
Microsoft Windows 10 Home [53]	2	145 €	6 meses	48 meses	18,125 €
Ubuntu 16.04 LTS [54]	1	0 €	6 meses	48 meses	0 €
Microsoft Office 365 [55]	1	99 €	6 meses	12 meses	49,5 €
Editor *Notepad++* [56]	1	0 €	6 meses	48 meses	0 €
GanttProject [57]	1	0 €	6 meses	48 meses	0 €
				Total	**85,75 €**

Tabla 106. Detalle de recursos software

El coste total del proyecto se muestra en la Tabla 107:

Concepto	Coste
Mano de obra	5760 €
Costes indirectos (15% mano de obra)	864 €
Recursos *hardware*	147,875 €
Recursos *software*	85,75 €
Total	**6857,625 €**

Tabla 107. Coste total del proyecto

10.2 Impacto

La solución se ha planteado e implementado en el ámbito de los juegos de *Atari 2600*. Sin embargo, la idea es adaptable y aplicable en otros ámbitos. Podría extenderse a otros videojuegos y serviría para construir *Non-Player Characters* (NPC), es decir, personajes que ejecutan por sí mismos tareas dentro del juego: atacar, defender, desplazarse, etc.

Otro ámbito que podría beneficiarse es la robótica. La solución desarrollada podría servir para que un robot resuelva tareas (desplazamientos, manipulación de objetos, etc.) reservadas hasta ahora a las personas.

A continuación, se hablará del impacto económico, social y ético que podría producirse si se emplease la solución propuesta en diversos ámbitos. El impacto sería similar al de aplicar cualquier otra técnica de Inteligencia Artificial.

- **Económico.**

  En el ámbito de la robótica, la idea propuesta podría contribuir a que los robots resuelvan tareas duras o monótonas para un trabajador, y sin necesidad de descansar. Las empresas podrían aprovecharse de estos avances para sustituir mano de obra humana por robótica. Las consecuencias de esto serían la pérdida de puestos de trabajo tradicionales, aunque podrían surgir nuevos empleos relacionados con la construcción y mantenimiento de robots. La eficiencia de los robots podría incrementar el margen de beneficios de las empresas.

  En el ámbito de los videojuegos, la idea propuesta podría contribuir a crear juegos más creíbles y desafiantes para los usuarios, lo que atraería a un mayor público y produciría un aumento de las ventas. Esto incrementaría la competitividad dentro del sector, y supondría la creación de nuevos puestos de trabajo.

- **Social.**

  En el ámbito de la robótica, la idea propuesta podría contribuir a que los robots aprendan a ejecutar tareas duras o a ayudar a personas en su vida cotidiana. Esto mejoraría su calidad de vida. Sin embargo, la falta de sentimientos y empatía de los robots podría frustrar a las personas que estén en contacto con ellos.

  En el ámbito de los videojuegos, como ya se ha comentado, la idea propuesta podría contribuir a crear juegos más desafiantes. Esto podría intensificar el problema de adicción a los videojuegos, especialmente entre la población joven.

- **Ético.**

  En el ámbito de la robótica, la autonomía de las máquinas para tomar decisiones podría ser un problema para la humanidad si las máquinas adquiriesen una inteligencia que superase los límites biológicos. Otra cuestión que se plantea es en quién recaería la responsabilidad en caso de que el robot cometiese, voluntaria o involuntariamente, algún incidente durante la ejecución de su tarea.

  En el ámbito de los videojuegos, se plantearía la cuestión de si merece la pena aumentar los beneficios empresariales desarrollando videojuegos cada vez más atractivos, a costa de fomentar la adicción en colectivos en riesgo.

11 Marco regulador

El trabajo desarrollado no utiliza información personal, no compromete la seguridad de personas u organizaciones y no conlleva responsabilidades profesionales. Ninguna ley es aplicable al proyecto.

El *software* sobre el que se basa la solución propuesta en el proyecto cuenta con licencias de *software* libre, ofrecen libertad para su modificación, copia, publicación, distribución y venta:

- La implementación del algoritmo DQN, procedente de un repositorio de *GitHub* [21], cuenta con licencia MIT [58].
- La implementación del algoritmo IW, procedente de un repositorio de *GitHub* [20], cuenta con licencia *GNU General Public License versión 2* (GPLv2) [59].

El *software* desarrollado en este proyecto, al ser una modificación de los dos mencionados anteriormente, también dispone de las mismas libertades. No existe una patente de la idea, ni de su ejecución.

Con respecto a las herramientas *software* empleadas:

- Ubuntu 16.04 LTS [54], *Notepad++* [56] y *GanttProject* [57] cuentan con licencia GPL.
- Windows 10 Home [53] y Microsoft Office 2016 [55] son *software* propietario con licencia de usuario final (EULA), y por tanto se han respetado las restricciones impuestas en cuanto a adquisición y distribución del *software*.

12 Conclusiones y comentarios

12.1 Conclusiones fundamentales

Los resultados de la experimentación sugieren que el nuevo agente es una buena alternativa a los enfoques tradicionales, pero no debe ser una sustitución a estos. Existen más de 50 videojuegos en el dominio de Atari, por lo que difícilmente una técnica destacará sobre el resto en todos ellos. Lo importante es que el agente sea competente en el mayor número de situaciones posibles. Si comparamos los resultados de *CombinationAgent* y *SearchAgent* bajo las mismas condiciones experimentales, el nuevo agente parece competente incluso en los videojuegos en los que se ve superado por *SearchAgent*, ya que las diferencias de puntuación en estos peores casos son muy reducidas.

La colaboración de dos técnicas tan dispares como DQN e IW parece ser un buen mecanismo para la toma de decisiones, y abre la puerta a la incorporación de otras técnicas de IA que han demostrado ser efectivas en el dominio de Atari como CGP, del caso de estudio sobre *genetic programming* [18].

En cuanto al tiempo de ejecución medio por *frame*, el nuevo agente ha demostrado ser tan veloz como *SearchAgent* debido a que el tiempo que invierte la red para determinar una acción es despreciable en comparación con el tiempo que necesita IW para realizar sus simulaciones.

Una desventaja de este agente es que no puede establecerse una configuración de parámetros universal para todos los videojuegos, por lo que sería necesario experimentar en cada uno de ellos para determinar los valores idóneos de parámetros como *ratio* o *limite_simulacion*.

Otro inconveniente es el entrenamiento de las redes, que es mejorable si lo comparamos con los resultados de *DeepMind* en su artículo sobre DQN [16]. Sin embargo, los resultados de la experimentación demuestran que unas redes poco inteligentes pueden ser de gran ayuda a IW para resolver partidas en determinados videojuegos. Si no existiesen restricciones que limitasen el aprendizaje de las redes, probablemente el entrenamiento mejoraría, así como las puntuaciones del agente *CombinationAgent*.

12.2 Opiniones personales

La temática del TFG, ligada al sector del entretenimiento, ha sido muy acertada para estudiar y desarrollar técnicas de Inteligencia Artificial porque ha sido clave para mantener la motivación durante los meses que ha durado el trabajo.

El proyecto ha sido un buen escenario para poner en práctica muchos conceptos aprendidos a lo largo del grado. Aunque el TFG gira entorno a la Inteligencia Artificial, para implementar el agente ha sido necesario usar una metodología de desarrollo

propia del área de la Ingeniería del *Software*. También ha sido una buena forma de poner en práctica las habilidades transversales como toma de decisiones, compromiso, flexibilidad, planificación de tareas, responsabilidad y trabajo bajo presión.

El análisis del estado del arte es una parte fundamental porque permite concretar los objetivos del proyecto. El estudio ha sido muy exhaustivo para sentar unas sólidas bases sobre las que construir el proyecto. Hay que tener en cuenta que algunos conceptos teóricos como *Deep Learning* se han visto de refilón durante el grado, otros ni siquiera (*IW* o *DQN*, por ejemplo) y otros muchos debían ser repasados (*aprendizaje por refuerzo*, *redes de neuronas*, *búsqueda y planificación*, etc.).

12.3 Dificultades y problemas encontrados

En cuanto al comienzo del proyecto:

- Fue difícil organizar las ideas sobre cómo se desarrollaría el proyecto porque el tema escogido era muy novedoso para mí, y fue necesario un periodo de adaptación y de investigación.

En cuanto a la etapa de desarrollo *software*:

- Fue imprescindible aprender C++ con mayor profundidad, porque hasta entonces nunca había tenido que interpretar o escribir un código en ese lenguaje (sí en C). Antes de empezar fue necesario un periodo de adaptación.
- Entender el *framework* ALE fue duro porque el código, por lo general, estaba pésimamente comentado y no disponía de una documentación clara sobre la arquitectura y las funcionalidades del *software*.

Con respecto a la experimentación:

- El entrenamiento de las redes fue especialmente complicado. La principal causa fue las limitaciones *hardware*. La implementación de DQN era incompatible con la GPU del ordenador, por lo que los entrenamientos se llevaron a cabo con la CPU, alargando con ello el tiempo de experimentación.
- También hubo limitaciones de memoria RAM que impidieron compaginar en el mismo ordenador la experimentación con el desarrollo de la documentación. Por ello, fue necesario pedir prestado un antiguo ordenador para realizar la memoria mientras se llevaba a cabo el entrenamiento en el otro ordenador.
- Todas las fases de la experimentación necesitaron muchos días de ejecución ininterrumpida. Cada entrenamiento de redes de DQN necesitó aproximadamente 1 día y fueron más de 10 los entrenamientos que se llevaron a cabo (entre definitivos y de prueba). En cuanto la experimentación con *CombinationAgent*, la gran cantidad de experimentos obligó al ordenador a ejecutar durante varios días, entre experimentos de prueba y definitivos.

- La depuración del código fue agobiante porque verificar su buen funcionamiento requería paciencia hasta que terminase de ejecutar. Aunque no fueron muchos los errores inesperados, cuando se producían obligaban a reiniciar la experimentación.

12.4 Trabajos futuros

Podría realizarse la misma experimentación con el agente *CombinationAgent*, pero sustituyendo IW por otros algoritmos de planificación que pueden encontrarse en el caso de estudio sobre IW [17]. Se escogió IW por ser objeto de estudio en dicho artículo y por su eficacia y eficiencia, pero existen muchos otros algoritmos.

Con más recursos *hardware*, podrían suprimirse las restricciones establecidas en las etapas 1 y 2 de la experimentación para mejorar el entrenamiento de las redes. Si se lograse la compatibilidad del código de DQN con una GPU, podría agilizarse enormemente el aprendizaje de las redes, consiguiendo redes mejor entrenadas en menos tiempo. Además, si se dispusiese de más tiempo, podría evaluarse el nuevo agente con el resto de videojuegos disponibles en el *framework* ALE.

Por último, los avances en el dominio de los videojuegos de Atari podrían servir para otros ámbitos de la informática como la robótica, como ya se ha comentado en el entorno socioeconómico.

13 Bibliografía

[1] N. J. Nilsson, «Preliminaries,» de *Introduction to Machine Learning: An Early Draft of Proposed Textbook*, Noviembre 1998, pp. 1-14.

[2] M. van Otterlo y M. Wiering, «Reinforcement Learning and Markov Decision Processes,» *Reinforcement Learning,* vol. 12, pp. 3-42, 2012.

[3] N. J. Nilsson, «Neural Networks,» de *Introduction to Machine Learning: An Early Draft of Proposed Textbook*, Noviembre 1998, pp. 35-62.

[4] J. Schmidhuber, «Deep Learning in Neural Networks: An Overview,» *Neural Networks,* vol. 61, pp. 85-117, Enero 2015.

[5] Y. LeCun y Y. Bengio, «Convolutional Networks for Images, Speech, and Time Series,» de *The Handbook of Brain Theory and Neural Networks*, 1998, pp. 255-258.

[6] S. Russell y P. Norvig, «Problem-solving,» de *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 2009, pp. 53-148.

[7] Y. Saez y P. Isasi, «Material de clase - Computación Biológica,» OCW UC3M, 2015. [En línea]. Available: http://ocw.uc3m.es/ingenieria-informatica/computacion-biologica/material-de-clase.

[8] «Atari,» [En línea]. Available: https://www.atari.com/corporate/about-atari-sa.

[9] F. «Neoteo - La historia de Atari,» 17 Diciembre 2007. [En línea]. Available: https://www.neoteo.com/la-historia-de-atari/.

[10] M. G. Bellemare, Y. Naddaf, J. Veness y M. Bowling, «Github, repositorio "Arcade-Learning-Environment",» 29 Diciembre 2017. [En línea]. Available: https://github.com/mgbellemare/Arcade-Learning-Environment.

[11] «Gym,» OpenAI, [En línea]. Available: https://gym.openai.com/.

[12] «Tensorflow,» [En línea]. Available: https://www.tensorflow.org/.

[13] «PyTorch,» [En línea]. Available: https://pytorch.org/.

[14] V. Mnih, K. Kavukvuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra y M. Riedmiller, «Playing Atari with Deep Reinforcement Learning,» de *Conference on Neural Information Processing Systems (NIPS)*, Lake Tahoe, Enero 2013.

[15] M. G. Bellemare, Y. Naddaf, J. Veness y M. Bowling, «The Arcade Learning Environment: An Evaluation Platform for General Agents,» *Journal of Artificial Intelligence Research,* vol. XLVII, nº 1, pp. 253-279, Mayo 2013.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg y D. Hassabis, «Human-level control through deep reinforcement learning,» *Nature,* vol. 518, nº 7540, pp. 529-533, 2015.

[17] N. Lipovetzky, M. Ramírez y H. Geffner, «Classical Planning with Simulators: Results on the Atari Video Games,» de *International Joint Conference on Artificial Intelligence (IJCAI)*, Buenos Aires, Julio 2015.

[18] D. G. Wilson, H. Luga, J. F. Miller y S. Cussat-Blanc, «Evolving simple programs for playing Atari games,» *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18),* pp. 229-236 , Julio 2018.

[19] «Pros y contras de la metodología en cascada,» OBS, [En línea]. Available: https://www.obs-edu.com/es/blog-project-management/metodologia-agile/pros-y-contras-de-la-metodologia-en-cascada.

[20] N. Lipovetzky, M. Ramirez y H. Geffner, «GitHub, repositorio "ALE-Atari-Width",» 3 Diciembre 2015. [En línea]. Available: https://github.com/miquelramirez/ALE-Atari-Width.

[21] devsisters, «GitHub, repositorio "DQN-tensorflow",» 28 Junio 2017. [En línea]. Available: https://github.com/devsisters/DQN-tensorflow.

[22] A. M. Turing, «Computing Machinery and Intelligence,» *Mind,* vol. LIX, nº 236 , pp. 433-460, Octubre 1950.

[23] S. E. Siwek, «Video Game in the 21st Century,» The Entertaiment Software Association, 2017.

[24] P. E. Hart, N. J. Nilsson y B. Raphael, «A Formal Basis for the Heuristic Determination of Minimum Cost Paths,» *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, nº 2, pp. 100-107, Julio 1968.

[25] M. O. Riedl y M. Young, «Narrative Planning: Balancing Plot and Character,» *Journal of Artificial Intelligence Research,* vol. 39, pp. 217-268, Septiembre 2010.

[26] N. Shaker, J. Togelius y M. J. Nelson, «Chapter 4: Fractals, noise and agents with applications to landscapes,» de *Procedural Content Generation in Games*, Springer, 2016, pp. 57-72.

[27] J. Togelius, N. Shaker y J. Dormans, «Chapter 5: Grammars and L-systems with applications to vegetation and levels,» de *Procedural Content Generation in Games*, Springer, 2016, pp. 73-98.

[28] J.-P. Kelly, A. Botea y S. Koenig, «Offline Planning with Hierarchical Task Networks in Video Games,» *AIIDE,* pp. 60-65, Octubre 2008.

[29] A. Johansson y P. Dell' Acqua, «Emotional Behavior Trees,» *IEEE Conference on Computational Intelligence and Games (CIG),* pp. 355-362, Septiembre 2012.

[30] C. J. C. H. Watkins y P. Dayan, «Q-learning,» *Machine Learning,* vol. 8, pp. 279-292, Mayo 1992.

[31] J. Fernández Bes, «Reinforcement Learning,» Noviembre 2012. [En línea]. Available: http://www.tsc.uc3m.es/~jesusfbes/MLG_RL.pdf.

[32] I. M. Galván, J. M. Valls, R. Aler y J. Huertas, «Material de clase - Redes de Neuronas Artificiales - Tema 1: Introducción a las redes de neuronas artificiales,» OCW UC3M, 2015. [En línea]. Available: http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas/transparencias/Tema1%20IntroduccionRN.pdf.

[33] Y. LeCun, Y. Bengio y G. Hinton, «Deep Learning,» *Nature,* vol. 521, pp. 436-444, Mayo 2015.

[34] I. M. Galván, J. M. Valls, R. Aler y J. Huertas, «Material de clase - Redes de Neuronas Artificiales - Tema 7: Introducción a deep learning,» OCW UC3M, 2015. [En línea]. Available: http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas/transparencias/Tema%207%20DeepLearning.pdf.

[35] R. E. Korf, «Depth-First Iterative-Deepening: An Optimal Admissible Tree Search,» *Artificial Intelligence,* vol. 27, pp. 97-109, 1985.

[36] «Museo de Informática de la República Argentina - Atari Pong,» [En línea]. Available: http://museodeinformatica.org.ar/consolas-de-juegos/atari/atari-pong/.

[37] D. Escandell, «Vandal - Los Orígenes de los Videojuegos - Atari VCS,» [En línea]. Available: https://vandal.elespanol.com/reportaje/los-origenes-de-los-videojuegos/4.

[38] D. Boris, «atarihq - Atari 2600 (System Specs),» [En línea]. Available: http://atarihq.com/danb/a2600.shtml.

[39] «Atarimania - Pong Sports,» [En línea]. Available: http://www.atarimania.com/game-atari-2600-vcs-pong-sports_16564.html.

[40] «Atarimania - Asteroids,» [En línea]. Available: http://www.atarimania.com/game-atari-2600-vcs-asteroids_8579.html.

[41] «Atarimania - Star Gunner,» [En línea]. Available: http://www.atarimania.com/game-atari-2600-vcs-stargunner_16921.html.

[42] «Atarimania - Montezuma's Revenge,» [En línea]. Available: http://www.atarimania.com/game-atari-2600-vcs-montezuma-s-revenge_7986.html.

[43] K. Dubovikov, «Towards Data Science: PyTorch vs TensorFlow - spotting the difference,» 20 Junio 2017. [En línea]. Available: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b.

[44] «skymind - Comparison of Frameworks,» [En línea]. Available: https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch.

[45] «Procesadores Intel Xeon,» Intel, [En línea]. Available: https://www.intel.la/content/www/xl/es/products/processors/xeon.html.

[46] S. Džeroski y B. Ženko, «Is Combining Classifiers with Stacking Better than Selecting the Best One?,» *Machine Learning,* vol. 54, pp. 255-273, Marzo 2004.

[47] JorgeCP7, «GitHub, repositorio "ALE-Atari-Combination",» 2018. [En línea]. Available: https://github.com/JorgeCP7/ALE-Atari-Combination.

[48] «4kepics,» [En línea]. Available: https://www.4kepics.com/atari-easter-egg-google-atari-games/.

[49] J. Ragan, «lakupo,» [En línea]. Available: https://www.lakupo.com/grblitz/space%20invaders%201.htm.

[50] «8-Bit Central,» [En línea]. Available: http://www.8-bitcentral.com/reviews/2600beamrider.html.

[51] A. Breder, «Gaga Games,» 24 Abril 2018. [En línea]. Available: http://www.gagagames.com.br/recordar-e-envelhecer-frostbite-atari-2600/.

[52] M. S. Zavia, «Gizmodo - Una IA aprende a hacer trampa en un juego de Atari con un fallo que nadie había encontrado hasta ahora,» 3 Enero 2018. [En línea]. Available: https://es.gizmodo.com/una-ia-aprende-a-hacer-trampa-en-un-juego-de-atari-con-1823425838.

[53] «Windows 10,» Microsoft, [En línea]. Available: https://www.microsoft.com/es-es/windows.

[54] «Ubuntu 16.04,» Canonical Ltd., [En línea]. Available: http://releases.ubuntu.com/16.04/.

[55] «Office,» Microsoft, [En línea]. Available: https://products.office.com/es-es/home.

[56] «Notepad++,» [En línea]. Available: https://notepad-plus-plus.org/.

[57] «GanttProject,» [En línea]. Available: https://www.ganttproject.biz/.

[58] «Open Source Iniciative - The MIT License,» [En línea]. Available: https://opensource.org/licenses/MIT.

[59] «GNU - Licencias,» [En línea]. Available: https://www.gnu.org/licenses/licenses.es.html.

[60] «Real Academa Española,» [En línea]. Available: http://www.rae.es/.

14 Glosario

14.1 Siglas

Siglas	Significado
ALE	*Arcade Learning Environment* (*framework* de desarrollo).
CGP	*Cartesian Genetic Programming* (técnica de computación evolutiva).
CNN	*Convolutional Neural Network*. *Red Neuronal Convolucional* en castellano.
DL	*Deep Learning*. *Aprendizaje Profundo* en castellano.
DQN	*Deep Q-Learning* (algoritmo).
ESA	*Entertainment Software Association*.
EULA	*End User License Agreement*.
GB	*Gigabyte* (unidad de medida de la información).
GNU	*GNU's Not Unix*.
GPL	*General Public License*.
GPU	*Graphics Processing Unit*. *Unidad de Procesamiento Gráfico* en castellano.
HTN	*Hierarchical Task Network* (algoritmo de planificación automática).
IA	*Inteligencia Artificial*.
IW	*Iterated Width* (algoritmo).
MIT	*Massachusetts Institute of Technology*.
ML	*Machine Learning*. *Aprendizaje Automático* en castellano.
MLP	*Multilayer Perceptron*. *Perceptrón Multicapa* en castellano.
NPC	*Non-Player Character*.
PDDL	*Planning Domain Definition Language.*
POO	*Programación Orientada a Objetos*.
RAM	*Random Access Memory* (memoria).
ReLU	*Rectifier Linear Unit* (función de activación).
RGB	*Red, Green, Blue* (modelo de color).
RL	*Reinforcement Learning. Aprendizaje por Refuerzo* en castellano.
ROM	*Read-Only Memory* (memoria).

Tabla 108. Acrónimos

14.2 Definiciones

Palabra	Definición
Adaline	Tipo de red de neuronas capaz de resolver tareas propias del aprendizaje automático, como clasificación o regresión.
Buffer	*Búfer* en castellano. Se trata de un espacio de memoria capaz de almacenar información de forma temporal.
Byte	Unidad de información que consta de 8 bits.
C++	Lenguaje de programación imperativo que representa una mejora del lenguaje C, incluyendo nuevas características como la programación orientada a objetos (POO).
DeepMind	Empresa británica dedicada a la Inteligencia Artificial.
Frame	*Fotograma* en castellano. Cada una de las imágenes que se suceden en un videojuego o película [60].
Framework	Se trata de un esquema que facilita el desarrollo de una aplicación. También denominado *entorno de trabajo* en castellano.
Fully-connected	En el contexto de las redes neuronales, se trata de una topología de red en la que cada neurona de una capa se conecta a todas las neuronas de la capa siguiente.
Hardware	*"Conjunto de aparatos de una computadora"*, [60].
Heurística	En el contexto de las ciencias de la computación, conocimiento parcial sobre un problema, que sirve de guía para la búsqueda de soluciones.
Joystick	*Palanca de mando* en castellano. Se trata de un periférico de entrada constituido por una palanca capaz de girar sobre un soporte.
Niebla de guerra	En el contexto de los videojuegos, se refiere a la niebla que oculta las zonas del mapa no descubiertas por el jugador.
Python	Lenguaje de programación interpretado que destaca por la legibilidad de su código y por la facilidad para implementar programas.
Software	*"Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora"*, RAE [60].
Trackball	*Bola de seguimiento* en castellano. Se trata de un dispositivo constituido por sensores capaces de detectar

Palabra	Definición
	la rotación de una bola, y que suele formar parte de algún periférico de entrada.

Tabla 109. Definiciones

15 Apéndices

15.1 ANEXO I. Ejemplo IW

A continuación, se presenta un ejemplo para ilustrar el concepto de novedad:

> Un problema con un espacio de estados de 2 variables (X_1, X_2), que pueden adoptar 2 valores: $\{0,1\}$.
> Un espacio de acciones arbitrario $\{a, b, c, d, e, f\}$

Tabla 110. Definición del problema

Se lleva a cabo una búsqueda desde un estado inicial (0, 0), utilizando $IW(1)$ e $IW(2)$:

- $IW(1)$

  En la Ilustración 68 se observa cómo se podan todos los nodos de profundidad 2 por tener novedades mayores a 1. Ninguno de los nodos podados es el primero en la búsqueda en hacer cierta una tupla de variables de tamaño 1.

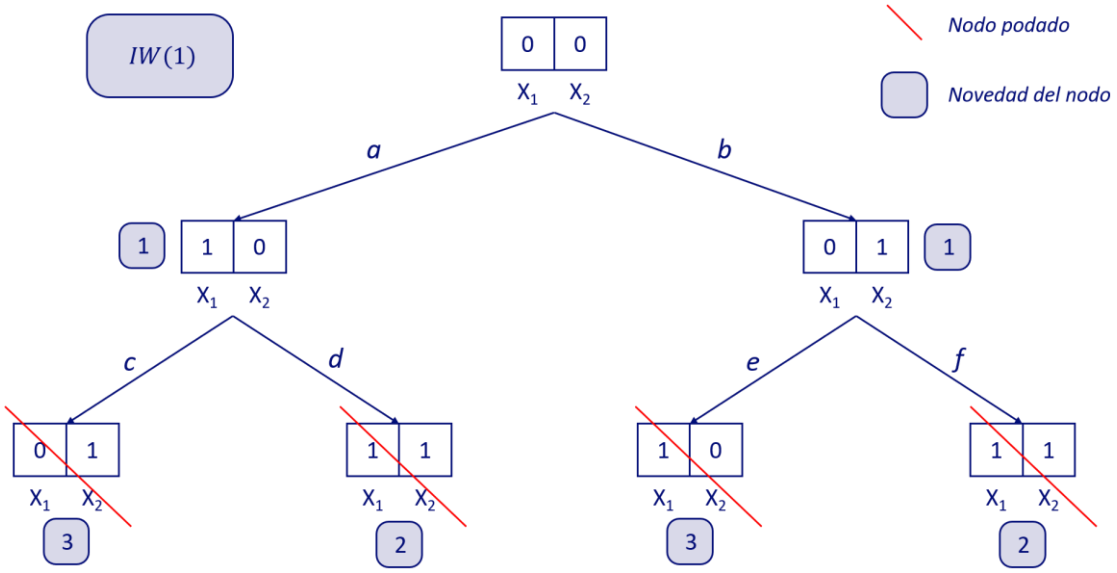


Ilustración 68. Ejemplo de IW(1)

- $IW(2)$

  En la Ilustración 69 se observa cómo se podan todos los nodos de profundidad 2, salvo el segundo nodo de profundidad 2 por no tener novedad mayor a 2. Ninguno de los nodos podados es el primero en la búsqueda en hacer cierta una tupla de variables de tamaño menor o igual a 2.

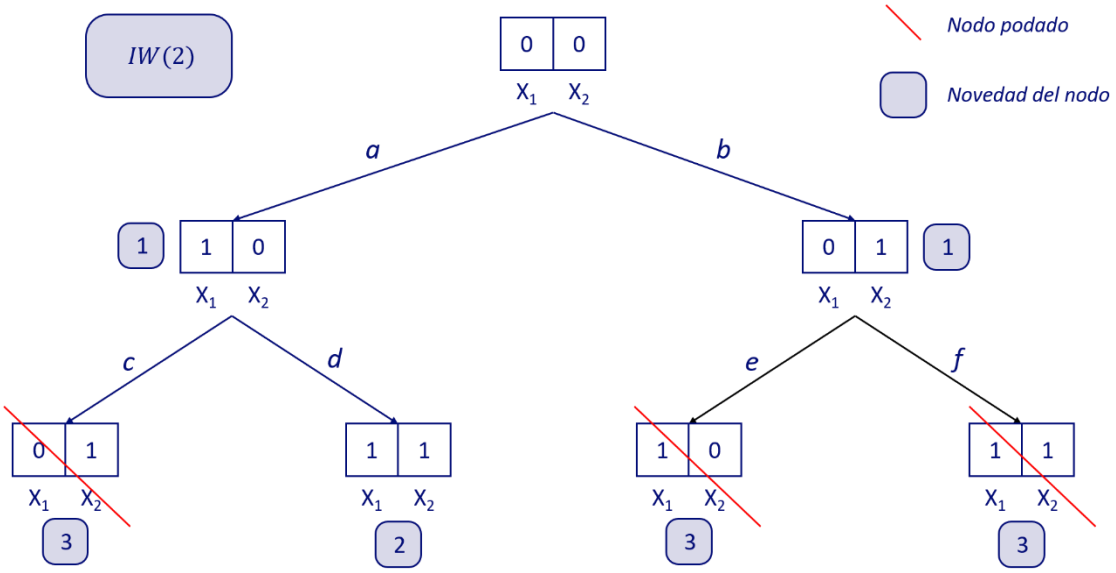


Ilustración 69. Ejemplo de IW(2)